

A Hybrid Quasi-Newton Projected-Gradient Method with Application to Lasso and Basis-Pursuit Denoise

E. van den Berg

IBM Watson, Yorktown Heights, NY, USA

November 18, 2016

Abstract

We propose a new algorithm for the optimization of convex functions over a polyhedral set in \mathbb{R}^n . The algorithm extends the spectral projected-gradient method with limited-memory BFGS iterates restricted to the present face whenever possible. We prove convergence of the algorithm under suitable conditions and apply the algorithm to solve the Lasso problem, and consequently, the basis-pursuit denoise problem through the root-finding framework proposed by van den Berg and Friedlander [SIAM Journal on Scientific Computing, 31(2), 2008]. The algorithm is especially well suited to simple domains and could also be used to solve bound-constrained problems as well as problems restricted to the simplex.

1 Introduction

In this paper we propose an algorithm for optimization problems of the form

$$\underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad x \in \mathcal{C}, \quad (1)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex, twice continuously differentiable function, and \mathcal{C} is a polyhedral set in \mathbb{R}^n . The main focus of the paper is the specialization and application of the framework to the Lasso problem [18]:

$$\underset{x}{\text{minimize}} \quad \frac{1}{2} \|Ax - b\|_2^2 \quad \text{subject to} \quad \|x\|_1 \leq \tau, \quad (\text{LS}_\tau)$$

where \mathcal{C} is a, possibly weighted, one-norm ball and $f(x) = \frac{1}{2} \|Ax - b\|_2^2$. The work in this paper was motivated by the need for an efficient and accurate solver for the Lasso subproblems appearing in the SPGL1 [2] solver for basis-pursuit denoise [11] problems of the form

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \frac{1}{2} \|Ax - b\|_2^2 \leq \sigma. \quad (\text{BP}_\sigma)$$

Both formulations are central to compressed sensing [9, 12] as a means of recovering exactly or approximately sparse vectors x_0 from linearly compressed and often noisy observations $b = Ax_0 + z$. In practice we may have a better idea about the noise level $\|z\|_2$, appearing as σ in the basis-pursuit denoise formulation, rather than the one-norm of the unknown signal x_0 , appearing as τ in Lasso. The (BP_σ) formulation is therefore often a more natural choice.

It was shown in [2] that basis-pursuit denoise and Lasso are connected through the Pareto curve

$$f(\tau) = \min_x \quad \|Ax - b\|_2^2 \quad \text{subject to} \quad \|x\|_1 \leq \tau,$$

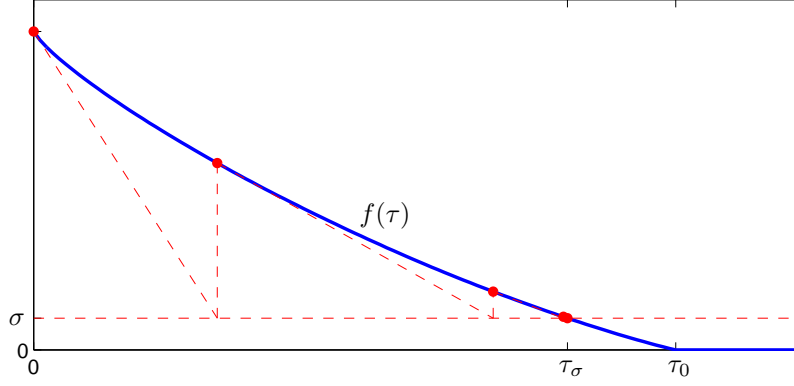


Figure 1: *Example of root finding on the Pareto curve $f(\tau)$.*

and that solving (BP_σ) can be reduced to finding the smallest τ for which the Lasso solution x_τ^* satisfies $\|Ax_\tau^* - b\| \leq \sigma$. Denoting by τ_σ this critical value of τ and assuming that b lies in the range space of A it was shown in [2] that the Pareto curve is convex and differentiable at all $\tau \in [0, \tau_0)$ with gradient $\|A^T r\|_\infty / \|r\|_2$ where r denotes the misfit $Ax_\tau^* - b$. Evaluation of both $f(\tau)$ and $f'(\tau)$ relies on the misfit r , which can be obtained by solving (LS_τ) . The SPGL1 solver proposed in [2] applies root finding on the Pareto curve, as illustrated in Figure 1, to solve $f(\tau) = \sigma$ and thereby reduce basis-pursuit denoise to a series of Lasso problems. In SPGL1 these subproblems are solved using the spectral projected-gradient (SPG) algorithm [7], which we discuss in more detail in Section 2. For certain problems it was found that SPG generates long sequences of iterates that all lie on the same face of the feasible set. This suggests an active-set type of method in which a quasi-Newton method, such as the limited-memory BFGS (L-BFGS) method [16], is used to minimize the problem restricted to the current face. Rather than carefully deciding when an active set has stabilized and then accurately solving over the active set before switching back to the global mode, we propose a hybrid algorithm that uses seamless and lightweight switching between the two methods. By doing so, we are able to take full advantage of the strengths of both methods, while avoiding possibly costly subproblem solves, or complicated heuristics that determine when to switch between the solvers.

1.1 Paper outline

In Section 2 we provide a concise background on the SPG and L-BFGS methods along with some of their theoretical properties. We then describe the proposed algorithm for the general problem formulation (1) in Section 3. In Section 4 we study the geometry of the constraints in the Lasso problem, and develop the tools needed for an efficient implementation of the framework for Lasso. Numerical experiments are provided in Section 5, followed by the conclusions in Section 6.

1.2 Notations and definitions

We use calligraphic capital letters for sets. Given any two set \mathcal{S}_1 and \mathcal{S}_2 , we write $\mathcal{S}_1 + \mathcal{S}_2$ for $\{x_1 + x_2 \mid x_1 \in \mathcal{S}_1, x_2 \in \mathcal{S}_2\}$, and likewise for $\mathcal{S}_1 - \mathcal{S}_2$. For a seeming lack of established terminology, we define the *difference hull* of a set \mathcal{S} , $\text{diff hull}(\mathcal{S})$, as the linear hull of differences $\{u_1 - u_2 \mid u_1, u_2 \in \mathcal{S}\}$. The difference hull can be seen as the linear subspace corresponding to the affine hull of \mathcal{S} translated to contain the origin. For any x in a polyhedral set \mathcal{C} , we define $\mathcal{F}(x)$ to be the unique face \mathcal{F} of \mathcal{C} for which $x \in \text{relint}(\mathcal{F})$; this may be \mathcal{C} itself. The normal cone of \mathcal{C} at x is given by $\mathcal{N}(x) := \{d \in \mathbb{R}^n \mid \mathcal{P}(x + d) = x\}$. The normal cone of a face \mathcal{F} is understood to

be $\mathcal{N}(x)$ for any $x \in \text{relint}(\mathcal{F})$. Orthogonal projection of any vector $v \in \mathbb{R}^n$ onto \mathcal{C} is defined as

$$\mathcal{P}(v) := \arg \min_x \|x - v\|_2 \quad \text{subject to} \quad x \in \mathcal{C}.$$

We define the *self-projection cone* of a face $\mathcal{F} = \mathcal{F}(x)$ as the closed and convex cone of directions $d \in \mathbb{R}^n$ such that there exists an $\epsilon > 0$ for which the projection of $x + \epsilon d$ lies on \mathcal{F} :

$$\text{self proj}(\mathcal{F}) = \mathcal{S}(\mathcal{F}(x)) := \{d \in \mathbb{R}^n \mid \exists \epsilon > 0 : \mathcal{F}[\mathcal{P}(x + \epsilon d)] = \mathcal{F}(x)\} = \mathcal{N}(x) + \text{diff hull}(\mathcal{F}(x)).$$

Note that $\mathcal{N}(x) \perp \text{diff hull}(\mathcal{F}(x))$; in fact, the difference hull of \mathcal{F} is the orthogonal complement of the linear hull of $\mathcal{N}(\mathcal{F})$. For any k -face \mathcal{F} of \mathcal{C} , $k \geq 1$, we denote by $\Phi(\mathcal{F}) \in \mathbb{R}^{n \times k}$ an arbitrary but fixed orthonormal basis for $\text{diff hull}(\mathcal{F})$. We will never use $\Phi(\mathcal{F})$ when \mathcal{F} is a vertex and therefore leave it undefined. We denote by e_i the i -th column of an identity matrix whose size is clear from the context. The proximation of a function f is defined as $\text{prox}_f(u) := \arg \min_x f(x) + \frac{1}{2}\|x - u\|_2^2$.

2 Background

2.1 The nonmonotone spectral projected-gradient method

The nonmonotone spectral projected-gradient method (SPG) was introduced by Birgin, Martínez, and Raydan [7] for problems of the form (1), with \mathcal{C} a closed convex set in \mathbb{R}^n , and $f : \mathbb{R}^n \rightarrow \mathbb{R}$ a function with continuous partial derivatives on an open set that contains \mathcal{C} . The algorithm is outlined in Algorithm 1, and it can be seen that the main step in each iteration is a line search along the curvilinear trajectory given by (see also [5]):

$$x(\alpha) = \mathcal{P}(x_i - \alpha \nabla f(x_i)), \quad \alpha \geq 0. \quad (2)$$

Two important modifications to the curvilinear projected-gradient method, made to help speed up convergence, were introduced in [7]. The first modification allows a limited level of nonmonotonicity in the objective value. Given $\mu, \gamma \in (0, 1)$, the Armijo-type line search starts with an initial step length α_i , and then finds the first nonnegative integer k such that

$$f(x(\mu^k \alpha_i)) \leq \max_{0 \leq j \leq \min\{i, M-1\}} \{f(x_{i-j})\} + \gamma(\nabla f(x_i))^T (x(\mu^k \alpha_i) - x_i). \quad (3)$$

The right-hand side of this condition ensures sufficient descend, but only with respect to the maximum of up to M of the most recent objective values. In case $M = 1$ this reduces to the standard Armijo line-search condition. The second modification is the use of the spectral step length, as proposed by Barzilai and Borwein [1]. Given $s = x_i - x_{i-1}$ and $y = \nabla f(x_i) - \nabla f(x_{i-1})$, the initial step length at iteration i is defined as

$$\alpha_i = \begin{cases} \max \left\{ \min \left\{ \frac{s^T s}{s^T y}, \alpha_{\max} \right\}, \alpha_{\min} \right\} & \text{if } s^T y > 0, \\ \alpha_{\max} & \text{otherwise,} \end{cases}$$

where $0 < \alpha_{\min} < \alpha_{\max}$ are fixed parameters. More information on the motivation behind this particular choice of step length can be found in [7, 14]. Under the conditions stated at the beginning of the section, it holds that any accumulation point x^* of the sequence $\{x_i\}$ is a constrained stationary point [7]; that is a point $x^* \in \mathcal{C}$ such that

$$\|\mathcal{P}(x^* - \nabla f(x^*)) - x^*\|_2 = 0. \quad (4)$$

In practice a relaxed version of (4) is used as a stopping criterion in Algorithm 1, along with other conditions.

```

Input:  $A, b, \tau, x_0$ 
Initialize  $i \leftarrow 0$ , choose  $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$ 
while not done do
    Compute  $g_i = \nabla f(x_i)$ 

    # Compute the Barzilai-Borwein scaling parameter
    if  $i > 0$  then
        Set  $s \leftarrow x_i - x_{i-1}$ ,  $y \leftarrow g_i - g_{i-1}$ 
        if  $s^T y > 0$  then
             $\alpha_i \leftarrow \max \left\{ \min \left\{ \frac{s^T s}{s^T y}, \alpha_{\max} \right\}, \alpha_{\min} \right\}$ 
        else
             $\alpha_i \leftarrow \alpha_{\max}$ 
        end
    end

    # Non-monotone curvilinear Armijo line-search
    Initialize  $k \leftarrow 0$ 
    while condition (3) is not satisfied do
         $k \leftarrow k + 1$ 
    end

    # Proceed to the next iteration
    Set  $x_{i+1} = x(\beta^k \alpha_i)$ 
    Update  $i \leftarrow i + 1$ 
end

```

Algorithm 1: *The nonmonotone spectral projected-gradient method (SPG).*

2.2 Limited-memory BFGS

The L-BFGS algorithm by Liu and Nocedal [16] is a popular quasi-Newton method for unconstrained minimization of smooth functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$\underset{x}{\text{minimize}} \quad f(x).$$

At each iteration, the algorithm constructs a positive definite approximation H_i of the inverse Hessian of f at x_i . This construction is based on an initial positive definite matrix H and $\hat{n} = \min\{i, N\}$ of the most recent vector pairs $\{s_{i-j}, y_{i-j}\}_{j=0}^{\hat{n}-1}$, with

$$s_j = x_j - x_{j-1}, \quad \text{and} \quad y_j = \nabla f(x_j) - \nabla f(x_{j-1}). \quad (5)$$

The iterates are of the form $x_{i+1} = x_i + \alpha_i d_i$, where the search direction d_i is given by

$$d_i = -H_i \cdot \nabla f(x_i), \quad (6)$$

and the step size α_i is chosen to satisfy the Wolfe conditions:

$$f(x_i + \alpha_i d_i) \leq f(x_i) + \gamma_1 \alpha_i (\nabla f(x_i))^T d_i, \quad (7a)$$

$$(\nabla f(x_i + \alpha_i d_i))^T d_i \geq \gamma_2 (\nabla f(x_i))^T d_i. \quad (7b)$$

Parameters γ_1 and γ_2 are chosen such that $0 < \gamma_1 < \frac{1}{2}$, and $\gamma_1 < \gamma_2 < 1$. For details on the structure of the inverse approximation H_i and efficient ways of evaluating the matrix-vector product in (6), see [16, 17].

2.2.1 Convergence results

For the analysis of the L-BFGS algorithm, Liu and Nocedal make the following assumptions [16]:

Assumption 2.1. *For a given starting point x_0 , we have that:*

- (1) *The objective function f is twice continuously differentiable;*
- (2) *The level set $\mathcal{D} := \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$ is convex; and*
- (3) *There exist positive constants μ_1 and μ_2 such that*

$$\mu_1 \|v\|^2 \leq v^T [\nabla^2 f(x)] v \leq \mu_2 \|v\|^2, \quad (8)$$

for all $x \in \mathcal{D}$ and $v \in \mathbb{R}^n$.

Under these conditions, and with some simplifications, they prove that

Theorem 2.2 (Liu and Nocedal [16]). *The L-BFGS algorithm generates a sequence $\{x_i\}$ that converges to the unique minimizer x_* in \mathcal{D} . Moreover, there exists a constant $c > 0$ such that*

$$f(x_{i+1}) - f(x^*) \leq (1 - c)(f(x_i) - f(x^*)). \quad (9)$$

3 Proposed algorithm

The proposed algorithm can be seen as a modification of the SPG method that allows the use of quasi-Newton steps over a currently active face. The basic idea is that whenever two successive iterates x_i and x_{i-1} lie on the same face, we can form or update a quadratic model of the objective function restricted to the face. To avoid unnecessary updates to the model and, indeed, to ensure convergence to a global optimizer, we require that $-g_i := -\nabla f(x_i)$ lies in the self-projection cone of $\mathcal{F}_i := \mathcal{F}(x_i)$. Whenever a model for the current face is available, the algorithm will attempt a quasi-Newton step that is restricted to the face and satisfies the Wolfe conditions (7). If the quasi-Newton step fails, or is otherwise abandoned, the algorithm simply falls back and takes a regular SPG step. After each step—regardless of the type—we again check the conditions required to update the quadratic model and initiate the quasi-Newton step:

$$\mathcal{F}(x_i) = \mathcal{F}(x_{i-1}) \quad \text{and} \quad -g_i \in \text{self proj}(\mathcal{F}_i). \quad (10)$$

If these conditions are not met, we discard the Hessian approximation used in the quadratic model, for example, by setting it to the empty set. Note that omitting the self-projection criterion from (10) could cause the algorithm to take repeated quasi-Newton iterations that converge to a minimum on the relative interior of the face that is not the global minimum.

3.1 Quasi-Newton over a face

One way of performing quasi-Newton over a face is by maintaining an inverse Hessian approximation using the update vectors in (5), and computing the search direction d_i using (6). However, this approach has some major disadvantages. First, we may have that $d_i \notin \text{diff hull}(\mathcal{F}_i)$, which means that $x_i + \alpha d_i \notin \mathcal{F}_i$ for all nonzero α . This could be partially solved by projection onto the face, but such a projected direction is no longer guaranteed to be a descent direction [6]. This too could be addressed by modifying the Hessian, but doing so would further complicate

```

Input:  $A, b, \tau, x_0$ 
Initialize  $H_0 = \emptyset, \mathcal{F}_0 = \mathcal{F}(x_0), g_0 = \nabla f(x_0)$ 
Set  $t \leftarrow 0$ , choose  $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$ 
while not done do
    # Quasi-Newton step on current face
    flagUpdated  $\leftarrow$  false
    if ( $H_t \neq \emptyset$ ) then
         $d_t = -\Phi_t H_t \Phi_t^T g_t$ 
        Perform Wolfe-type line-search on  $x(\gamma) := x + \gamma d$  over  $\{\gamma \mid x(\gamma) \in \mathcal{F}_t\}$ 
        if (line search successful) then
            Reset objective function history
             $x_{t+1} = x_t + \gamma_t d_t$ 
            flagUpdated  $\leftarrow$  true
        end
    end
    # Projected-gradient step
    if (flagUpdated = false) then
        Compute Barzilai-Borwein scaling parameter
        Nonmonotone curvilinear Armijo line-search along  $x(\gamma) := \mathcal{P}(x_t - \gamma g_t)$ 
         $x_{t+1} = x(\gamma_t)$ 
    end
     $t \leftarrow t + 1, g_t = \nabla f(x_t), \mathcal{F}_t = \mathcal{F}(x_t)$ 
    # Update the quadratic model of the current face
    if ( $\mathcal{F}_t = \mathcal{F}_{t-1}$  and  $-g_t \in \text{self proj}(\mathcal{F}_t)$ ) then
        if ( $H_{t-1} = \emptyset$ ) then
            Initialize  $H_{t-1} \leftarrow \mu I$ 
            Determine orthonormal basis  $\Phi_t = \Phi(\mathcal{F}_t)$ 
        else
            Set  $\Phi_t = \Phi_{t-1}$ 
        end
        Set  $H_t$  as L-BFGS update to  $H_{t-1}$  using  $s_t = \Phi_t^T(x_t - x_{t-1})$  and  $y_t = \Phi_t^T(g_t - g_{t-1})$ 
    else
         $H_t = \emptyset$ 
    end
end

```

Algorithm 2: Outline of the proposed hybrid quasi-Newton projected-gradient method.

the algorithm. A second disadvantage is that we maintain the inverse Hessian approximation for the ambient space, which typically has a much higher dimension than the current face and may therefore not be very accurate along $\text{aff}(\mathcal{F}_i)$.

The solution of the above problem is straightforward: we simply work with a representation for the function restricted to $\text{aff}(\mathcal{F}_i)$. Let \mathcal{F}_i be a k -dimensional face with $k > 0$. Then we can find an orthonormal basis $\Phi := \Phi(\mathcal{F}_i) \in \mathbb{R}^{n \times k}$ whose span coincides with $\text{diff hull}(\mathcal{F}_i)$. Using Φ we can write any point $x \in \mathcal{F}_i$ as $x = v + \Phi c$, where $v \in \mathbb{R}^n$ is an arbitrary but fixed point in \mathcal{F}_i , and $c \in \mathbb{R}^k$ is a coefficient in the lower-dimensional space. The function $\hat{f} : \mathbb{R}^k \rightarrow \mathbb{R}$, which restricts f to the current face, is then given by $\hat{f}(c) = f(v + \Phi c)$. The idea then is to form the inverse Hessian approximation over \hat{f} , and use it to obtain a search direction $\hat{d} \in \mathbb{R}^k$, which can then be mapped back to the ambient space for the actual line search. In particular, we can form

the approximate inverse Hessian $H_i \in \mathbb{R}^{k \times k}$ by updating an initial positive definite H using

$$s_i = \Phi^T(x_i - x_{i-1}) \quad \text{and} \quad y_i = \Phi^T(g_i - g_{i-1}). \quad (11)$$

In order to obtain the search direction we first compute $\nabla \hat{f}(\Phi^T(x_i - v)) = \Phi^T g_i$ by projecting the gradient g_i onto the lower-dimensional space. We then apply the inverse Hessian followed by back projection, giving:

$$d_i = \Phi H_i \Phi^T(-g_i).$$

The resulting vector clearly lies in $\text{diff hull}(\mathcal{F}_i)$ and we therefore have that $x_i + \alpha d_i \in \mathcal{F}_i$ for all step sizes $\alpha \in [0, \alpha_{\text{bnd}}]$, for some $\alpha_{\text{bnd}} > 0$, possibly with $\alpha_{\text{bnd}} = +\infty$. Since the line search is done in the ambient dimension, we try to find a step size α within the above range, such that the original Wolfe conditions (7) are met. For the line search we could start with a unit step length, whenever $\alpha_{\text{bnd}} \geq 1$, or we could try $\alpha = \alpha_{\text{bnd}}$ first to encourage exploring lower-dimensional faces, provided of course that $\alpha_{\text{bnd}} < \infty$. If no suitable step length can be found, or a certain maximum number of trial steps is taken, we abandon the quasi-Newton step and take a spectral projected-gradient step instead. As a final remark, note that condition (10) should never be met for vertices, since that would imply not only that $x_i = x_{i-1}$, but also that $-g_{i-1} \in \text{self proj}(\mathcal{F}_{i-1}) = \mathcal{N}(\mathcal{F}_{i-1})$, which means that the optimality condition given in (4) would already have been satisfied at x_{k-1} .

3.2 Convergence

For the convergence analysis of Algorithm 2 we rely on the results in [7] and [16], and add a step in the algorithm that resets the objective-value history used by SPG after each series of successful quasi-Newton iterations to ensure that any subsequent iteration has a lower objective value. We use the following assumptions, which are somewhat more restrictive than those in the aforementioned two papers (see, for example, Assumption 2.1):

Assumption 3.1. *We assume that*

- (1) *The objective function f is convex, twice continuously differentiable, and bounded below;*
- (2) *There exist constants $0 < \mu_1 \leq \mu_2 < \infty$ such that for all $x, v \in \mathbb{R}^n$:*

$$\mu_1 \|v\|_2^2 \leq v^T H(x) v \leq \mu_2 \|v\|_2^2. \quad (12)$$

Under these assumptions, we have the following result:

Theorem 3.2. *Let $f(x)$ satisfy Assumptions 3.1 and let $x_0 \in \mathcal{C}$. Then the sequence $\{x_t\}$ generated by Algorithm 2 converges to the unique minimizer of (1).*

Proof. Assumption 3.1 ensures the existence of a unique minimizer x^* to (1), which satisfies

$$-g(x^*) \in \mathcal{N}(x^*).$$

If there exists a finite t for which $x_t = x^*$, we are done. Suppose, therefore that $x_t \neq x^*$ for all t . We consider two cases. First, if there are finitely many quasi-Newton steps, there must a \bar{t} such that all iterations $t > \bar{t}$ are of the projected gradient type. In this case the result follows directly from the analysis in [7]. Second, consider the case where there are infinitely many quasi-Newton steps. It can be seen that each quasi-Newton step works towards minimizing the objective over the affine hull of the current face \mathcal{F} :

$$\underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad x \in \text{aff}(\mathcal{F}). \quad (13)$$

For any such step it follows from the analysis in Liu and Nocedal [16] (with minor modification to the number of update vectors available, and working in the lower-dimensional and unconstrained representation based on Φ) that there exists a constant $c > 0$ such that the quasi-Newton step satisfies

$$f(x_{t+1}) - f(x_{\mathcal{F}}^*) \leq (1 - c)(f(x_t) - f(x_{\mathcal{F}}^*)), \quad (14)$$

where $x_{\mathcal{F}}^*$ denotes the minimizer of (13). Because the history of the M most recent objective values is reset after each successful quasi-Newton step, any intermediate projected-gradient step will not increase the objective. Based on this, Lemma 3.3 below, shows that the number of quasi-Newton iterates on any \mathcal{F} that does not contain x^* is finite. By polyhedrality of the domain, the number of faces itself is bounded, and we must therefore take infinitely many iterations on at least one face that contains x^* . Repeated application of (14) then shows that the objective value converges to $f(x_{\mathcal{F}}^*)$. Finally, it follows from Assumption 3.1 that $\{x_t\}$ converges to x^* . \square

Lemma 3.3. *Let \mathcal{F} be a face of \mathcal{C} such that $x^* \notin \mathcal{F}$. Then the number of quasi-Newton steps on \mathcal{F} taken by Algorithm 2 is finite.*

Proof. Let $x_{\mathcal{F}}^*$ be the solution to (13), and denote by $x_{[j]}$ and $x_{[j]+1}$ the starting, respectively ending, point for the j -th quasi-Newton step on \mathcal{F} . It can be seen that

$$f(x_{[j]}) - f(x_{\mathcal{F}}^*) \leq f(x_{[j-1]+1}) - f(x_{\mathcal{F}}^*) \leq (1 - c)(f(x_{[j-1]}) - f(x_{\mathcal{F}}^*)), \quad (15)$$

for $j \geq 2$. This holds since any intermediate quasi-Newton iteration can only reduce the objective, and likewise for projected-gradient steps, as a consequence of resetting the function-value history.

We consider two cases. First assume that $x_{\mathcal{F}}^* \notin \mathcal{F}$. Let \bar{f} be the minimum of $f(x)$ over $x \in \mathcal{F}$. Repeated application of (15) gives

$$f(x_{[j]+1}) - f(x_{\mathcal{F}}^*) \leq (1 - c)^j (f(x_{[1]}) - f(x_{\mathcal{F}}^*)). \quad (16)$$

For sufficiently large, but finite j , the right-hand side in (16) must fall below $\bar{f} - f(x_{\mathcal{F}}^*)$, which is strictly positive. Since every successful quasi-Newton step results in a vector $x_{[j]+1} \in \mathcal{F}$ by construction, it follows that the number of quasi-Newton iterates on \mathcal{F} must be bounded.

For the second case, assume that $x_{\mathcal{F}}^* \in \mathcal{F}$. Because optimization is done over $\text{aff}(\mathcal{F})$, it holds that $-g(x_{\mathcal{F}}^*) \perp \text{diff hull}(\mathcal{F})$. For $-g(x_{\mathcal{F}}^*) \in \text{self proj}(\mathcal{F})$, we must therefore have $-g(x_{\mathcal{F}}^*) \in \mathcal{N}(x_{\mathcal{F}}^*)$, but this cannot be the case since it would imply that $x_{\mathcal{F}}^*$ is a global minimizer. (The same holds when $x_{\mathcal{F}}^*$ lies on a lower-dimensional surface on the boundary of \mathcal{F} .) Since f is continuously differentiable by assumption, it follows that $-g(x) \notin \text{self proj}(\mathcal{F})$ for all points $x \in \mathcal{F}$ sufficiently close to $x_{\mathcal{F}}^*$. Assumption 3.1 then allows us to define a sufficiently close neighborhood as the level set $f(x) \leq \bar{f}$ over $x \in \mathcal{F}$, where $\bar{f} > f(x_{\mathcal{F}}^*)$. Applying the same argument we used above shows that the right-hand side of (12) again falls below $\bar{f} - f(x_{\mathcal{F}}^*)$ for sufficiently large j . Once this happens all following iterates $x_t \in \mathcal{F}$ must have $f(x_t) \leq \bar{f}$. Since the self-projection cone condition $-g(x) \in \text{self proj}(\mathcal{F})$ does not hold at these points, no more quasi-Newton steps are taken on \mathcal{F} . \square

A similar analysis holds when the spectral projected-gradient method is replaced by another convergent algorithm, provided that the iterates do not exceed the initial objective value.

4 Application to Lasso

The proposed algorithm depends on a number of operations on the constraint set. In particular, it has to determine in which face the current iterate lies, check membership of the self-projection cone, and determine an orthonormal basis for the current face. For the algorithm to be of practical use, the constraint set therefore needs to be simple enough to allow efficient evaluation of these operations. As this work was motivated by improving the Lasso problem, we focus on the weighted one-norm ball (which for unit weights is also known as the cross-polytope or n -octahedron [15]):

$$\mathcal{C}_{w,1} = \{x \in \mathbb{R}^n \mid \|x\|_{w,1} \leq \tau\},$$

where $\|x\|_{w,1} := \sum_i w_i |x_i|$ positive w_i . The proposed framework also applies naturally to bound or simplex constrained problems, but these are outside the scope of this paper.

The objective function we consider throughout this section is

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2 + \frac{\mu}{2}\|x\|_2^2 + c^T x, \quad (17)$$

which can also be written in the form $\frac{1}{2}\|Ax - b\|_2^2 + c^T x$, with A and b appropriately redefined. The benefit of having an objective function of the form (17) is that it permits closed-form expressions for step lengths satisfying certain conditions. In the remainder of this section we discuss practical considerations for the line-search conditions and look at the specific structure and properties of the set $\mathcal{C}_{w,1}$.

4.1 Line search

For most objective functions the line search is done by evaluating $f(\mathcal{P}(x + \alpha d))$ or $f(x + \alpha d)$ for a series of α values until all required conditions, such as Armijo and Wolfe, are satisfied. The objective function in (17) has closed-form solutions for some of the problems arising in the line search, thereby allowing us to simplify the algorithms and improve their performance.

4.1.1 Optimal unconstrained step size

As a start we look at the step length that minimizes the objective along $f(x + \alpha d)$:

$$\alpha_{\text{opt}} = \arg \min_{\alpha} f(x + \alpha d)$$

Differentiating f with respect to α and equating to zero leads to the following expression:

$$\alpha_{\text{opt}} = -\frac{(A^T r + \mu x + c)^T d}{\|Ad\|_2^2 + \mu\|d\|_2^2},$$

with $r = Ax - b$. When $\mu = 0$ and $c = 0$ this reduces to $\alpha_{\text{opt}} = -r^T Ad / \|Ad\|_2^2$.

4.1.2 Wolfe line search conditions

The maximum step length for which the Armijo condition (7a) is satisfied can be found by expanding the terms and simplifying. Doing so gives the following bound:

$$\alpha \leq \alpha_{\text{max}} = 2(1 - \gamma_1)\alpha_{\text{opt}}.$$

Likewise, the gradient condition (7b) reduces to

$$\alpha \geq \alpha_{\text{min}} = (1 - \gamma_2)\alpha_{\text{opt}}.$$

The derivations of these quantities are given in Sections B.1 and B.2 for completeness.

4.1.3 Projection arc

Line search in gradient projection methods is often done by backtracking from a single projection

$$p(\alpha) = x + \alpha(\mathcal{P}_C(x + d)),$$

with $\alpha \in [0, 1]$, or by search over the projection arc

$$p(\alpha) = \mathcal{P}_C(x + \alpha d),$$

with $\alpha \geq 0$. The trajectory of the first method depends strongly on the scaling of d and is more likely to generate points on the interior of the feasible set. The second method is invariant to the scaling of d and better captures the structure of the domain, but can also be more expensive computationally. The theorem below shows that the projection arc for polyhedral sets is piecewise linear and continuous, with a finite number of segments. In Sections 4.2.5 we provide an efficient algorithm for generating the successive line segments of the projection arc over $\mathcal{C}_{w,1}$. Combined with the closed-form solution of the optimum along each segment this gives an efficient and reliable algorithm for doing the line search along the entire projection arc.

Theorem 4.1. *Let \mathcal{C} be a convex polyhedron in \mathbb{R}^n . Then for any $x, d \in \mathbb{R}^n$, the projection trajectory $p(\alpha) := \mathcal{P}_C(x + \alpha d)$ is piecewise linear and continuous with a finite number of segments.*

Proof. The result is trivial for $d = 0$ so we assume $d \neq 0$. Let \mathcal{F}_i , $i = 1, \dots, N$ be the faces of \mathcal{C} . Then we partition the ambient space \mathbb{R}^n into convex polyhedral regions $\mathcal{R}_i = \mathcal{F}_i + \mathcal{N}(\mathcal{F}_i)$, where $\mathcal{N}(\mathcal{F}_i)$ is the normal cone to face \mathcal{F}_i and $+$ denotes the Minkowski sum (see Figure 2 for an illustration). The line $x + \alpha d$ intersects the boundary of each region for at most two values of α . Because of the one-to-one correspondence of regions and faces, it follows that the number of such breakpoints must be finite. Given two successive breakpoints α_k and α_{k+1} (possibly $\pm\infty$) corresponding to the intersection with some region \mathcal{R}_i . We first show that the projection trajectory $p(\alpha)$ for $\alpha_k \leq \alpha \leq \alpha_{k+1}$ is linear. For regions generated by a vertex it holds that all points on the line segment project to the vertex, thus giving a constant trajectory, which is trivially linear, and it remains to show linearity for higher dimensional faces. Let $v = x + \alpha d$ with $\alpha \in [\alpha_1, \alpha_2]$. Then $v = \mathcal{P}_C(v) + (v - \mathcal{P}_C(v))$, where the first and second terms lie respectively in \mathcal{F}_i and $\mathcal{N}(\mathcal{F}_i)$. Let $Q = \Phi(\mathcal{F})$ be an orthonormal basis for the difference hull of \mathcal{F}_i and choose any $u \in \mathcal{F}$ then

$$\begin{aligned} v - u &= \mathcal{P}_C(v) - u + (v - \mathcal{P}_C(v)) \\ QQ^T(v - u) &= QQ^T(\mathcal{P}_C(v) - u) + QQ^T(v - \mathcal{P}_C(v)) \\ QQ^T(v - u) &= \mathcal{P}_C(v) - u \\ \mathcal{P}_C(v) &= u + QQ^T(v - u) \end{aligned}$$

For the projection trajectory of the entire segment we therefore find

$$\begin{aligned} p(\alpha) &= u + QQ^T((x + \alpha d) - u) \\ &= (u + QQ^T(x - u)) + \alpha QQ^T d, \end{aligned}$$

which is linear in α , as desired. Because the space is partitioned, each finite breakpoint is simultaneously the end point of the line segment in one region and the starting point of the line segment in another region. Each value of α corresponds to one point $x + \alpha d$, which has a unique projection, thereby showing the continuity of the trajectory and completing the proof. \square

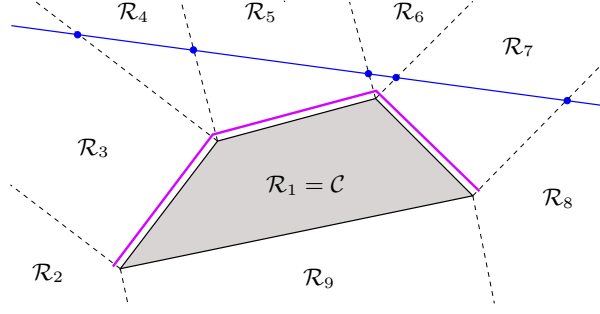


Figure 2: Illustration of a polytope \mathcal{C} and the associated partition of \mathbb{R}^2 into nine regions. The solid blue line (extended to infinity) intersects regions \mathcal{R}_2 through \mathcal{R}_8 and projects onto the corresponding faces, shown by the solid purple line, which is offset slightly from \mathcal{C} for clarity.

4.2 Properties of the weighted one-norm ball

4.2.1 Facial structure

The weighted one-norm ball of radius τ is the convex hull of vertices $\{\pm\tau/w_i \cdot e_i\}_i$. Every proper k -face \mathcal{F} of the weighted one-norm ball $\mathcal{C}_{w,1}$ can be written as the convex hull of $\{\sigma_i/w_i \cdot e_i\}_{i \in \mathcal{I}}$, where \mathcal{I} is a subset of $\{1, \dots, n\}$ with cardinality $k + 1$, and $\sigma_i \in \{-1, +1\}$. Throughout this section we assume that $\tau > 0$.

Given an $x \in \mathcal{C}$ we can determine $\mathcal{F}(x)$ as follows. First, we need to check whether $\|x\|_{w,1} < \tau$, in which case $\mathcal{F}(x) = \mathcal{C}$. Otherwise, x lies on a proper face, which can be uniquely characterized by the sign vector $\text{sgn}(x)$ whose i -th entry is given by $\text{sgn}(x_i)$. Determining $\mathcal{F}(x)$ and checking equality of faces can therefore be done in $\mathcal{O}(n)$ time.

4.2.2 Projection onto the feasible set

Projection onto the weighted one-norm ball is discussed in [3] and is based on the solution of the prox function

$$x_\lambda(u) = \text{prox}_{\lambda\|\cdot\|_{w,1}}(u) := \arg \min_x \frac{1}{2}\|x - u\|_2^2 + \lambda\|x\|_{w,1} = \text{sign}(u) \cdot [|u| - \lambda w]_+, \quad (18)$$

where $[\cdot]_+ = \max(0, \cdot)$, and the absolute value, sign function, and multiplication in the right-hand side are evaluated elementwise. Projection onto $\mathcal{C}_{w,1}$ then amounts to finding the smallest $\lambda \geq 0$ for which $\|x_\lambda(u)\|_{w,1} \leq \tau$. The entries in $x_\lambda(u)$, and therefore $\|x_\lambda(u)\|_{w,1}$, are continuous and piecewise linear in λ with break points occurring at $\lambda = |u_i|/w_i$. We can obtain an $\mathcal{O}(n \log n)$ algorithm that finds the optimal λ and subsequent projection by sorting the break points [3]. This can be reduced to an expected $\mathcal{O}(n)$ algorithm [13] by avoiding the explicit sorting step.

4.2.3 Self-projection cone of a face

Given $x \in \mathcal{C}_{w,1}$ and search direction $d \in \mathbb{R}^n$, we want to know if $d \in \text{self proj}(\mathcal{F}(x))$. When $\|x\|_{w,1} < \tau$ it follows that x lies in the interior of $\mathcal{C}_{w,1}$ meaning that $\mathcal{F}(x) = \mathcal{C}_{w,1}$ and $d \in \text{self proj}(\mathcal{C}_{w,1}) = \mathbb{R}^n$, trivially. For $\|x\|_{w,1} = \tau$, consider the support $\mathcal{I} = \{i \mid x_i \neq 0\}$. Because the entries on the support are bounded away from zero by definition and the soft-thresholding parameter λ is initially linear in α it follows that the support of $x(\alpha) = \mathcal{P}(x + \alpha D)$ includes \mathcal{I} for all sufficiently small α . For d to be in the self-projection cone we therefore need to show that

(1) $x + \alpha d$ does not move into the polytope, and (2) that the support does not increase. It can be verified that the first condition is satisfied if and only if

$$\sum_{i \in \mathcal{I}} \text{sign}(x_i) d_i w_i + \sum_{i \notin \mathcal{I}} |d_i| w_i \geq 0. \quad (19)$$

For the second condition to be satisfied we require for all $i \notin \mathcal{I}$ and sufficiently small α that the absolute value of entry remains less than or equal to the threshold value, namely $\alpha |d_i| \leq w_i \lambda(\alpha)$. When the support remains the same we find $\lambda(\alpha)$ by solving

$$\sum_{i \in \mathcal{I}} w_i (|x_i + \alpha d_i| - w_i \lambda(\alpha)) = \tau, \quad \text{which gives} \quad \lambda(\alpha) = \alpha \cdot \frac{\sum_{i \in \mathcal{I}} w_i \text{sign}(x_i) d_i}{\sum_{j \in \mathcal{I}} w_j^2},$$

after writing $|x_i + \alpha d_i| = |x_i| + \text{sign}(x_i) d_i$ and recalling that $\|x\|_{w,1} = \tau$. A necessary (and sufficient) condition for the support to remain the same is therefore that

$$\max_{i \notin \mathcal{I}} |d_i|/w_i \leq \frac{\sum_{i \in \mathcal{I}} w_i \text{sign}(x_i) d_i}{\sum_{i \in \mathcal{I}} w_i^2}. \quad (20)$$

Summarizing the above we have:

Theorem 4.2. *Given $x \in \mathcal{C}_{w,1}$ with support $\mathcal{I} = \{i \mid x_i \neq 0\}$, then $d \in \text{self proj}(\mathcal{F}(x))$ if and only if $\|x\|_{w,1} < \tau$, or $\|x\|_{w,1} = \tau$ and*

$$\sum_{i \in \mathcal{I}} \text{sign}(x_i) d_i w_i + \sum_{i \notin \mathcal{I}} |d_i| w_i \geq 0, \quad \text{and} \quad \max_{i \notin \mathcal{I}} |d_i|/w_i \leq \frac{\sum_{i \in \mathcal{I}} w_i \text{sign}(x_i) d_i}{\sum_{i \in \mathcal{I}} w_i^2}.$$

4.2.4 Orthogonal basis for a face

For the construction of a quadratic approximation of objective function f over a face \mathcal{F} , we require an orthogonal basis Φ for $\text{diff hull}(\mathcal{F})$. For simplicity, consider the facet of the unit cross polytope lying in the positive orthant in \mathbb{R}^3 . In other words, consider the unit simplex given by $\text{conv}\{e_1, e_2, e_3\}$. A first vector for the basis can then be obtained by normalizing $e_2 - e_1$ to have unit norm. A second vector orthogonal to the first can be obtained by connecting the point halfway on the line segment $e_1 - e_2$ to e_3 , that is, $e_3 - (e_1 + e_2)/2$, followed again by normalization. This can be generalized, and for a general k -simplex we find $(k+1) \times k$ basis $Q = [q_1, q_2, \dots, q_k]$ with

$$q_j = (e_{j+1} - \frac{1}{j} \sum_{i=1}^j e_i) / \sqrt{1 + 1/j}.$$

In other words

$$Q_{i,j} = \begin{cases} -\sqrt{1/(j^2 + j)} & i \leq j \\ \sqrt{j/(j+1)} & i = j+1 \\ 0 & \text{otherwise.} \end{cases}$$

It can be seen that the above procedure amounts to taking a QR factorization of the $(k+1) \times k$ matrix $[-e, I]^T$ of differences between the first vertex and all others, and discarding the last column in Q , whose entries are all equal to $1/\sqrt{n}$. The special structure of Q allows us to

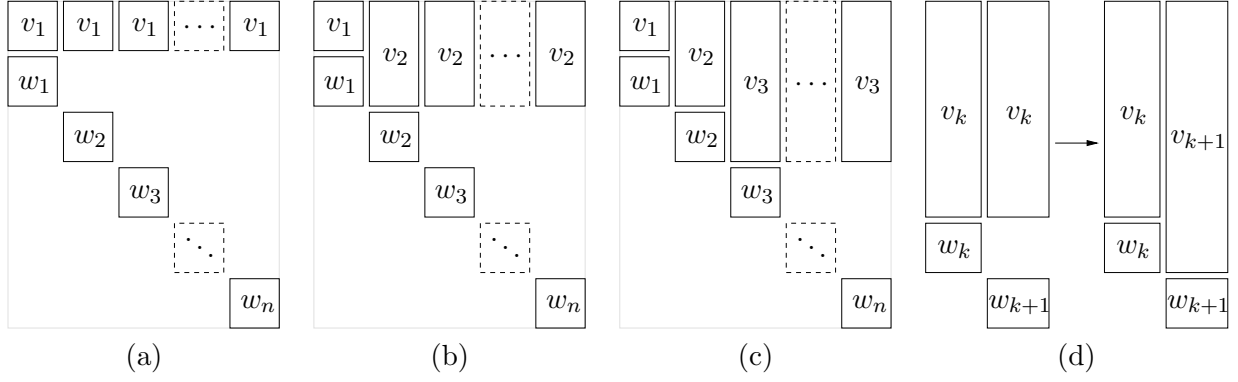


Figure 3: *Stages of the orthogonalization process.*

evaluate matrix-vector products with Q itself and its transpose in $\mathcal{O}(k)$ time, without having to form the matrix explicitly. For the general case, let $\mathcal{F} = \mathcal{F}(x)$. For the case where $\mathcal{F} = \mathcal{C}$ no projection is needed and we can simply choose $\Phi = I$. Otherwise, let $\mathcal{I} = \{i \mid x_i \neq 0\}$ denote the support of x . The desired basis can then be obtained by first restricting the vector to its support and then normalizing the sign pattern, thus giving:

$$\Phi = I_{\mathcal{I}} \cdot \text{diag}(\text{sgn}(x_{\mathcal{I}})) \cdot Q.$$

Matrix-vector products with Φ can be evaluated in $\mathcal{O}(n)$ time, again without forming the matrix

Generic weighted one-norm ball For the weighted one-norm ball we consider a face given by $\text{conv}(w_0 e_1, w_1 e_2, \dots, w_n e_n)$, with nonzero weights w_0 to w_n . (Throughout this paragraph it is more convenient to work with a weight vector whose elements are the inverse of the weights appearing in the weighted one norm; the actual vertices of the weighted one norm ball are $\pm w_i^{-1} e_i$, not $\pm w_i e_i$.) We would again like to obtain an orthonormal basis corresponding to the face. This can be done by applying QR factorization to the matrix of differences between the vertices, as illustrated in Figure 3(a) with $v_1 = -w_0$. The two operations in this process are projecting out the contributions of all subsequent columns and normalizing the columns to unit norm. We do not normalize until the very end but do keep track of the squared two norm of the completed columns. Given vectors a and b we obtain the component in b orthogonal to a by evaluating $b - \frac{\langle a, b \rangle}{\langle a, a \rangle} a$. In the first step of the factorization (we are interested only in Q) we orthogonalize with respect to the first column a . The inner product of each column with a is identical and equal to $\alpha_1 = \langle v_1, v_1 \rangle = \|v_1\|_2^2$. Using this we also compute the squared two norm of the first column as $\gamma_1 = \alpha_1 + w_1^2$. After the sweep with the first column we are left with the matrix shown in Figure 3(b) where

$$v_2 = \begin{bmatrix} v_1 - \frac{\alpha_1}{\gamma_1} v_1 \\ -\frac{\alpha_1}{\gamma_1} w_1 \end{bmatrix} = \begin{bmatrix} \frac{\gamma_1 - \alpha_1}{\gamma_1} v_1 \\ -\frac{\alpha_1}{\gamma_1} w_1 \end{bmatrix} = \begin{bmatrix} \frac{w_1^2}{\gamma_1} v_1 \\ -\frac{\alpha_1}{\gamma_1} w_1 \end{bmatrix}.$$

The next step is to sweep with the updated second column. For this we compute the inner product with the remaining columns and itself, yielding $\alpha_2 = \|v_2\|_2^2$ and $\gamma_2 = \alpha_2 + w_2^2$, respectively. After this sweep we arrive at the matrix given in Figure 3(c). Proceeding like this we successively sweep with each of the column until we are done. Consider now the sweep with some column k ,

illustrated in Figure 3(d). Given $\alpha_k = \|v_k\|_2^2$ and $\gamma_k = \alpha_k + w_k^2$ we find

$$v_{k+1} = \begin{bmatrix} v_k - \frac{\alpha_k}{\gamma_k} v_k \\ -\frac{\alpha_k}{\gamma_k} w_k \end{bmatrix} = \begin{bmatrix} \frac{w_k^2}{\gamma_k} v_k \\ -\frac{\alpha_k}{\gamma_k} w_k \end{bmatrix},$$

from which we derive recurrence relations

$$\alpha_{k+1} = \left(\frac{w_k^2}{\gamma_k}\right)^2 \|v_k\|_2^2 + \left(\frac{\alpha_k}{\gamma_k}\right)^2 w_k^2 = \frac{w_k^2 w_k^2}{\gamma_k^2} \alpha_k + \frac{\alpha_k^2}{\gamma_k^2} w_k^2 = \alpha_k w_k^2 \frac{w_k^2 + \alpha_k}{\gamma_k^2} = \frac{\alpha_k w_k^2}{\gamma_k}, \quad (21)$$

and $\gamma_{k+1} = \alpha_{k+1} + w_{k+1}^2$. With α_1 and γ_1 as given above, this allows us to compute all α and γ values. Ultimately we are interested in the final orthonormal Q matrix. Defining scaling factors

$$\mu_{i,j} = \prod_{k=i}^j \frac{w_k^2}{\gamma_k} \quad \text{for } 1 \leq i \leq j \leq n, \quad (22)$$

as well as factors $u_i = -\alpha_i/\gamma_i$ for $1 \leq i \leq n$ and $u_0 := -1$, it can be found based on the structure of the v vectors that

$$Q = \text{diag}(w) \cdot \begin{bmatrix} u_0 & \mu_{1,1}u_0 & \mu_{1,2}u_0 & \mu_{1,3}u_0 & \cdots & \mu_{1,n-1}u_0 \\ 1 & u_1 & \mu_{2,2}u_1 & \mu_{2,3}u_1 & \cdots & \mu_{2,n-1}u_1 \\ & 1 & u_2 & \mu_{3,3}u_2 & \cdots & \mu_{3,n-1}u_2 \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & 1 & u_{n-2} & \mu_{n-1,n-1}u_{n-2} \\ & & & & 1 & u_{n-1} \\ & & & & & 1 \end{bmatrix} \cdot \text{diag}(1/\sqrt{\gamma}).$$

Multiplication with this matrix and its transpose may still seem expensive but we now show how the structure enables $\mathcal{O}(n)$ algorithms for both operations. Multiplication with the diagonal matrices is trivial so we focus only on multiplication with the central part of the matrix. Looking at a small example we can decompose this matrix as

$$\begin{bmatrix} u_0 & \mu_{1,1}u_0 & \mu_{1,2}u_0 \\ 1 & u_1 & \mu_{2,2}u_1 \\ & 1 & u_2 \\ & & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix} + \text{diag}(u) \begin{bmatrix} 1 & \mu_{1,1} & \mu_{1,2} \\ & 1 & \mu_{2,2} \\ & & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix} + \text{diag}(u) \begin{bmatrix} M \\ 0 \end{bmatrix}$$

The key part is multiplication with the last matrix M . To evaluate $y = Mv$ we initialize $y_3 = v_3$ and then work upwards. Direct evaluation gives $y_2 = v_2 + \mu_{2,2}v_3$, which can be rewritten as $y_2 = v_2 + \mu_{2,2}y_2$. A pattern emerges when looking at the computation of y_1 :

$$\begin{aligned} y_1 &= v_1 + \mu_{1,1}v_2 + \mu_{1,2}v_3 \\ &= v_1 + \mu_{1,1}(v_2 + \mu_{2,2}v_3) \\ &= v_1 + \mu_{1,1}y_2, \end{aligned}$$

where $\mu_{1,2} = \mu_{1,1}\mu_{2,2}$, or more generally $\mu_{i,k} = \mu_{i,j}\mu_{j+1,k}$ for $i \leq j \leq k$, follows from the definition of μ in (22). Given $y_n = v_n$, we therefore obtain the recurrence $y_k = v_k + \mu_{k,k}y_{k+1}$, which allows us to evaluate $y = Mv$ in linear time. With v appropriately redefined we now look at $y = M^T v$:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & & \\ \mu_{1,1} & 1 & \\ \mu_{1,2} & \mu_{2,2} & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}.$$

Data: Weight vector $w = [w_1, \dots, w_n]$

Result: Vectors α , γ , u , and μ

Initialize $\alpha_1 = w_1^2$, $u_1 = -1$

for $k = 1$ to $n - 1$ **do**

$$\gamma_k = \alpha_k + w_{k+1}^2$$

$$\mu_k = w_{k+1}^2 / \gamma_k$$

$$u_{k+1} = -\alpha_k / \gamma_k$$

$$\alpha_{k+1} = \alpha_k \mu_k$$

end

Algorithm 3: Initialization for multiplication with the orthogonal basis for a face of the weighted one-norm ball.

Data: Vectors γ , u , w , μ and $v \in \mathbb{R}^{n-1}$

Result: Vector $y = Qv$

Initialize $s = v_{n-1} / \sqrt{\gamma_{n-1}}$, $t = 0$

$$y_n = w_n s$$

for $k = n - 1$ down to 2 **do**

$$t \leftarrow \mu_k t + s$$

$$s \leftarrow v_{k-1} / \sqrt{\gamma_{k-1}}$$

$$y_k = w_k (u_k t + s)$$

end

$$y_1 = w_1 u_1 (\mu_1 t + s)$$

Algorithm 4: Multiplication with orthogonal basis Q for a face of the weighted one-norm ball: $y = Qv$.

Data: Vectors γ , u , w , μ and $v \in \mathbb{R}^n$

Result: Vector $y = Q^T v$

$$t = w_1 v_1 u_1$$

$$s = w_2 v_2$$

$$y_1 = (t + s) / \sqrt{\gamma_1}$$

for $k = 2$ to $n - 1$ **do**

$$t \leftarrow \mu_{k-1} t + u_k s$$

$$s \leftarrow w_{k+1} v_{k+1}$$

$$y_k = (t + s) / \sqrt{\gamma_k}$$

end

Algorithm 5: Multiplication with orthogonal basis Q for a face of the weighted one-norm ball: $y = Q^T v$.

Starting with $y_1 = v_1$ we find $y_2 = \mu_{1,1} y_1 + v_2$ and $y_3 = \mu_{2,2} y_2 + v_3$, using $\mu_{1,2} = \mu_{1,1} \mu_{2,2}$. This gives the recurrence $y_{k+1} = \mu_{k,k} y_k + v_{k+1}$. We summarize the initialization and multiplication with Q and Q^T in Algorithms 3–5. Note that these algorithms use a different indexing scheme for a convenient implementation. For practical implementations we can precompute and store $1/\sqrt{\gamma_k}$ instead of γ_k and avoid storing α since it is not used during the evaluation of matrix-vector products. Alternatively, we can reduce the memory footprint at the cost of increased computation by storing only α and re-computing μ_k , u_k , and γ_k whenever they are needed.

4.2.5 Generation of the projection arc

In this section we consider the computation of the projection arc $p(\alpha) = \mathcal{P}(x(\alpha))$ of half line $x(\alpha) = s + \alpha d$ with $\alpha \geq 0$. We allow any starting point $s \in \mathbb{R}^n$, even though $s \in \mathcal{C}_{w,1}$ always holds for our application. From Section 4.1.3 we know that the projection arc is piecewise linear with discrete break points at $\alpha = \alpha_i$. As illustrated in Figure 4 there are three types of break points: (1) the support reduces and we move to a lower dimensional face (α_1 and α_5); (2) the support increases and we move to a higher dimensional face (α_2); and (3) we intersect the polytope boundary (α_3 and α_4). The algorithm for computing the projection arc thus proceeds as follows. Starting with $\alpha_0 = 0$ we compute at each iteration $i \geq 0$ the minimal $\alpha_{i+1} > \alpha_i$ for which one of the events occurs. Once this is done we compute $x_{(i)}$ and, based on the type of event, determine the corresponding $p_{(i)}$. The algorithm completes when none of the three events happens for α exceeding the current value. We now show how the next α for each event type is computed.

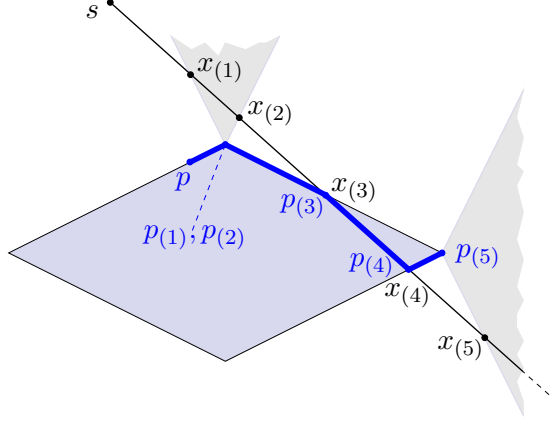


Figure 4: Illustration of half line $x(\alpha) = s + \alpha d$ (black) with its projection arc (thick blue) on a weighted one-norm ball. The break points on the original line and their corresponding projection are indicated by $x_{(i)}$ and $p_{(i)}$, respectively. At the intersections with the domain we have $p_{(3)} = x_{(3)}$ and $p_{(4)} = x_{(4)}$. The light gray wedges indicate the normal cones at two of the vertices.

Intersection with the boundary. In order to determine intersections with the boundary we need to keep track of $\kappa(\alpha) = \|x(\alpha)\|_{w,1}$. This function is linear with break points occurring whenever one of the elements in x crosses zero. Let $\mathcal{K} = \{j \mid s_j d_j < 0\}$ denote the set of indices that will at some point cross zero, and define $r_j = |d_j|$ when $j \in \mathcal{K}$ and $r_j = -|d_j|$ otherwise. We maintain the directional derivative of $\kappa(\alpha)$ with respect to increasing α as

$$\rho = \sum_{j \notin \mathcal{K}} w_j |d_j| - \sum_{j \in \mathcal{K}} w_j |d_j| = \sum_j r_j w_j. \quad (23)$$

Whenever some x_j reaches zero we remove j from the set \mathcal{K} , set r_j to $-r_j$, and update ρ to the value $\rho + 2w_j |d_j|$. To deal with these updates we add a fourth type of event corresponding to zero crossings. These happen at α values $-s_j/d_j$ for $j \in \mathcal{K}$, which that can be pre-computed and sorted at the beginning of the algorithm. At the beginning of iteration i we are given the current slope ρ and have

$$\kappa(\alpha_i + \delta) = \kappa(\alpha_i) + \delta \rho,$$

for limited $\delta \geq 0$. Solving for the next boundary intersection gives $\delta = +(\tau - \kappa(\alpha_i))/\rho$ and $\alpha = \alpha_i + \delta$. Whenever a zero crossing happens before this value of α the algorithm will encounter that event first, update ρ accordingly, and recompute δ in the next iteration. If $\alpha \leq \alpha_i$ we omit the boundary crossing event from consideration for the current iteration. Regardless of the event type we update $\kappa(\alpha_{i+1}) = \kappa(\alpha_i) + (\alpha_{i+1} - \alpha_i)\rho$. In the case of a boundary crossing (there can at most be two such events) we set $\lambda = 0$ and $\kappa = \tau$.

Changes in the support. Given $x = x_{(i)}$ with norm $\kappa = \|x\|_{w,1}$ and projection $p = \mathcal{P}(x)$ we want to find the smallest δ such that the support of $\mathcal{P}(x(\alpha_i + \delta))$ differs from $\mathcal{I} := \{i \mid p_i \neq 0\}$. We assume without loss of generality that either $\kappa > \tau$, or $\kappa = \tau$ with corresponding rate of change $\rho > 0$, otherwise we either move inside the polytope or along one of the faces, in which case the next event is guaranteed to either be a zero crossing or a boundary intersection. Many of the equations in this section, such as $|x_i + \delta d_i| = |x_i| + \delta r_i$, are valid only for sufficiently small δ and break down whenever there is a sign change or an intersection with the boundary of $\mathcal{C}_{w,1}$. We nevertheless use all equations as if they hold for all δ . The rationale for this is that we are interested in the first event. If the first event is a change in support then clearly there were no

Data: Sets \mathcal{I} and \mathcal{J}
Result: Set \mathcal{I}'
Set $a = \sum_{i \in \mathcal{I}} w_i r_i$, $b = \sum_{i \in \mathcal{I}} w_i^2$
Set $a_j = w_j r_j$, $b_j = w_j^2$ for all $j \in \mathcal{J}$
Initialize $\mathcal{I}' = \mathcal{I}$
while ($\mathcal{J} \neq \emptyset$) **do**
 Find j such that $a_j/p_j \geq a_k/p_k$ for all $k \in \mathcal{J}$
 Set $a \leftarrow a + a_j$, $b \leftarrow b + b_j$
 Set $\mathcal{J} = \{j \in \mathcal{J} \setminus \{j\} \mid a_j/b_j > a/b\}$
end

Algorithm 6: Selection of additions to the support given candidate set \mathcal{J} .

sign changes or boundary intersections before that point, which means that all equations used to compute δ were valid. Otherwise we do have a sign change or boundary intersection, in which case we ignore the incorrect δ value for the support change.

We now consider the point $x' = x(\alpha_i + \delta)$. For the support of the corresponding projection to remain the same we must have that the threshold parameter $\lambda'(\delta)$ computed based on the index set \mathcal{I} be consistent with x' . From the projection operator it follows that

$$\begin{aligned} \tau &= \sum_{i \in \mathcal{I}} w_i (|x_i + \delta d_i| - \lambda'(\delta) w_i) = \sum_{i \in \mathcal{I}} w_i (|x_i| + \delta r_i - \lambda'(\delta) w_i) \\ &= \kappa + \delta \sum_{i \in \mathcal{I}} w_i r_i - \lambda'(\delta) \sum_{i \in \mathcal{I}} w_i^2, \end{aligned}$$

from which we find

$$\lambda'(\delta) = \frac{(\kappa - \tau) + \delta \sum_{i \in \mathcal{I}} w_i r_i}{\sum_{i \in \mathcal{I}} w_i^2} = \lambda + \delta \frac{\sum_{i \in \mathcal{I}} w_i r_i}{\sum_{i \in \mathcal{I}} w_i^2} = \lambda + \delta \mu, \quad (24)$$

where λ is the threshold parameter for x and μ is the directional derivative with respect to δ . The resulting threshold parameter $\lambda'(\delta)$ is then consistent with x' if only if $\{i \mid |x'_i| > w_i \lambda'(\delta)\} = \mathcal{I}$.

Additions to the support. When considering additions to the support we assume, in addition to zero crossings and boundary intersections, that there are no events corresponding to variables leaving the support. A necessary condition for a variable $j \notin \mathcal{I}$ to enter the support is that $|x'_j| > w_j \lambda'(\delta)$, or equivalently $|x_j| + \delta r_j > w_j(\lambda + \delta \mu)$. From the definition of the support we have $|x_j| \leq w_j \lambda$, and for a variable to enter it must therefore hold that $r_j > w_j \mu$, or equivalently $r_j/w_j > \mu$. In this case, the value of δ at which variable j is about to enter is given by $\delta_j = (|x_j| - \lambda w_j) / (\mu w_j - r_j) \geq 0$, otherwise we set $\delta_j = +\infty$. The smallest δ for which an addition to the support is about to happen, if any, is then given by $\delta = \min_{j \notin \mathcal{I}} \delta_j$ with the set of variables staged to enter given by $\mathcal{J} = \{j \notin \mathcal{I} \mid \delta_j = \delta\}$. Provided that no event occurs before this point, at least one of the variables in \mathcal{J} will enter. As it does, it changes the rate of change in λ , which may mean that some of the other variables in \mathcal{J} never actually enter the support. As such, care needs to be taken in determining which variables enter and which do not; otherwise the same variable may repeatedly enter and leave the support, causing the algorithm to cycle forever. The following stage of the algorithm iteratively constructs the desired new support set \mathcal{I}' , which is the largest subset of $\mathcal{I} \cup \mathcal{J}$ such that $j \in \mathcal{J}$ is in \mathcal{I}' if and only if $r_j > w_j \mu'$ with

$$\mu' = \frac{\sum_{i \in \mathcal{I}'} w_i r_i}{\sum_{j \in \mathcal{I}'} w_j^2}. \quad (25)$$

Data: Vectors s, d
Result: Break points of the projection arc
Set $\mathcal{K} = \{j \mid x_j d_j < 0\}$ and compute r according to (23)
Initialize $\alpha_0 = 0, i = 0, \kappa_0 = \|s\|_{w,1}$
while true **do**
 Determine the next event α_{i+1}
 Return if there is no next event

 Update $\kappa_{i+1} = \kappa_i + (\alpha_{i+1} - \alpha_i)r$
 If zero crossing of index j : update $r \leftarrow r + 2|d_j|$
 If boundary crossing: set $\kappa_{i+1} \leftarrow \tau, \lambda_{i+1} = 0$.
end

Algorithm 7: *Outline of the general algorithm for computing all break points of the projection arc.*

The new set \mathcal{I}' is formed iteratively starting from \mathcal{I} by successively adding more elements from \mathcal{J} until it satisfies (25). Starting with $\mathcal{I}' = \mathcal{I}$ we define

$$a = \sum_{i \in \mathcal{I}} w_i r_i, \quad b = \sum_{i \in \mathcal{I}} w_i^2, \quad \text{and} \quad a_j = w_j r_j, \quad b_j = w_j^2 \text{ for } j \in \mathcal{J}.$$

This allows us to rewrite $r_j > w_j \mu$ as $a_j/b_j > \mu = a/b$. Given any $p_1/q_1 \prec p_2/q_2$ with $q_1, q_2 > 0$ and relational operator $\prec \in \{=, <, \leq\}$ it can be verified that

$$\frac{p_1}{q_1} \prec \frac{p_1 + p_2}{q_1 + q_2} \prec \frac{p_2}{q_2}. \quad (26)$$

Let $j \in \mathcal{J}$ be such that $a_j/b_j \geq a_k/b_k$ for all $k \in \mathcal{J}$. We show that j must be member of \mathcal{I}' . Assume by contradiction that $j \notin \mathcal{I}'$. By repeated application of (26) we find that

$$\frac{a}{b} < \frac{a + \sum_{i \in \mathcal{I}' \setminus \mathcal{I}} a_i}{b + \sum_{i \in \mathcal{I}' \setminus \mathcal{I}} b_i} = \mu' < \frac{\sum_{i \in \mathcal{I}' \setminus \mathcal{I}} a_i}{\sum_{i \in \mathcal{I}' \setminus \mathcal{I}} b_i} \leq \frac{a_j}{b_j}.$$

This shows that $r_j/w_j > \mu'$, which contradicts $j \notin \mathcal{I}'$. Given that j must be part of \mathcal{I}' we can add j to \mathcal{I}' and update a and b to $a + a_j$ and $b + b_j$, respectively. After this we remove element j from \mathcal{J} as well as all elements k for which $a_k/b_k \leq a/b$. This process is iterated until $\mathcal{J} = \emptyset$, at which point we have the final \mathcal{I}' . This algorithm is summarized in Algorithm 6.

Removal from the support. For the removal of items from the support, we start by finding the smallest δ such that $|x'_i| = w_i \lambda'(\delta)$ for some $i \in \mathcal{I}$, if any. Next, we define the set \mathcal{J} of entries staged to leave the support as the set of all entries $i \in \mathcal{I}$ for which the above equality holds for the given δ . Removal from the support causes the rate of change in λ to increase (see Section A for more details). This means that all entries staged in \mathcal{J} leave the support, and therefore that no further filtering is needed.

4.2.6 Line search along the projection arc

Given the set of break points α_k and the corresponding changes to the support we can perform a line search along the piecewise linear projection arc. When restricting all relevant vectors to

the support for a given segment, we can write $p(\alpha) = s + \alpha d - \lambda(\alpha)v$, where v is a vector of sign values for soft-thresholding. For the objective value we need $Ap(\alpha)$ for $\alpha_k \leq \alpha \leq \alpha_{k+1}$, which can then be written as $Ax(\alpha) = As + \alpha Ad - \lambda(\alpha)Av$. Once we have the three matrix-vector products with A , the objective function can easily be converted into a quadratic function in α by evaluating the appropriate inner products of these vectors. As the support or signs change we need to update the products As , Ad , and Av by adding or subtracting multiples of the required columns of A . Assuming that A is explicitly available or that the columns of A can be extracted in $\mathcal{O}(m)$ time, each update takes $\mathcal{O}(m)$ time. The objective function over each segment is quadratic and the minimum within the segment is therefore easily determined. For the evaluation of the overall computational complexity we need to know the maximum number of segments that a projection arc can have, or likewise, the maximum number of faces that a line can project onto. In Appendix A we prove the following result:

Theorem 4.3. *The projection of the line $x + \alpha d$ onto a (weighted) one-norm ball in \mathbb{R}^n is piecewise linear with at most $4n - 1$ segments. For every $n \geq 1$ there exist parameters x , d , and weights w for which this bound is achieved.*

Combined with this maximum number of possible segments possible, it follows that the line search can be done in $\mathcal{O}(mn)$ time. (In practice it may be necessary to recompute entirely the three matrix-vector products at regular intervals to avoid numerical issues.) For weighted one-norm balls the procedure requires slightly more bookkeeping but is otherwise the same.

4.2.7 Maximum step length along a face

Given a feasible search direction d it is useful to know the maximum α for which $x + \alpha d \in \mathcal{C}_{w,1}$. When x lies in the interior of $\mathcal{C}_{w,1}$ or when (19) is violated and $x + \alpha d$ moves into the interior, we need to compute the first intersection with the boundary. The procedure for doing this was described earlier in Section 4.2.5. When x lies on a proper face of $\mathcal{C}_{w,1}$ and d moves along the face or onto a higher dimensional face, the maximum step length is determined by the first element to reach zero:

$$\alpha_{\max} = \min_{i: x_i d_i < 0} -x_i / d_i.$$

4.3 Stopping criteria

We now look at stopping criteria for optimizing $f(x)$ as defined in (17) over the weighted one-norm ball. A common stopping criterion for problem of this type is to look at the relative norm of the projected gradient:

$$\rho(x) := \frac{\|\mathcal{P}_{\mathcal{C}}(x - \nabla f(x)) - x\|_2}{\max\{1, \|\nabla f(x)\|\}},$$

which is zero if and only if x is optimal. In addition to this we can look at the relative duality gap, which we define as the difference δ between $f(x)$ and any dual feasible objective, divided by $\max\{1, f(x)\}$.

For the derivation of the dual problem we follow [2, 3] and rewrite the original problem as:

$$\underset{x, r}{\text{minimize}} \quad \frac{1}{2} \|r\|_2^2 + c^T x + \frac{\mu}{2} \|x\|_2^2 \quad \text{subject to} \quad Ax + r - b = 0, \quad \|x\|_{w,1} \leq \tau.$$

The dual of this problem is given by

$$\underset{y, \lambda}{\text{maximize}} \quad \mathcal{L}(y, \nu) \quad \text{subject to} \quad \lambda \geq 0,$$

where the Lagrange dual function \mathcal{L} is given by

$$\begin{aligned}
\mathcal{L}(y, \lambda) &:= \inf_{x, r} \left\{ \frac{1}{2} \|r\|_2^2 + c^T x + \frac{\mu}{2} \|x\|_2^2 - y^T (Ax + r - b) + \lambda (\|x\|_{w,1} - \tau) \right\} \\
&= y^T b - \tau \lambda + \inf_r \left\{ \frac{1}{2} \|r\|_2^2 - y^T r \right\} + \inf_x \left\{ (c - A^T y)^T x + \frac{\mu}{2} \|x\|_2^2 + \lambda \|x\|_{w,1} \right\} \\
&= y^T b - \tau \lambda - \frac{1}{2} \|y\|_2^2 + \inf_x \left\{ (c - A^T y)^T x + \frac{\mu}{2} \|x\|_2^2 + \lambda \|x\|_{w,1} \right\}. \tag{27}
\end{aligned}$$

Here, the infimum over r is solved by equating the gradient to zero, giving $y = r$ and $y^T r = \|y\|_2^2$. For the infimum over x we consider two cases, based on the value of μ .

Dual when $\mu = 0$. With $c = 0$ this is exactly to the formulation considered in [3], and with minor changes it can be shown that

$$\inf_x \{ (c - A^T y)^T x + \lambda \|x\|_{w,1} \} = \begin{cases} 0 & \|A^T y - c\|_{\frac{1}{w}, \infty} \leq \lambda \\ -\infty & \text{otherwise.} \end{cases}$$

From this we then obtain the dual problem:

$$\begin{aligned}
&\underset{y, \lambda \geq 0}{\text{maximize}} && y^T b - \tau \lambda - \frac{1}{2} \|y\|_2^2 \quad \text{subject to} \quad \|A^T y - c\|_{\frac{1}{w}, \infty} \leq \lambda. \tag{28}
\end{aligned}$$

As a dual-feasible point we can choose $y = r$. For any given y it can be verified that choosing $\lambda = \|A^T y - c\|_{\frac{1}{w}, \infty}$ always gives the largest dual objective value. Given x and the corresponding residual $r = b - Ax$ we therefore obtain the following duality gap:

$$\delta = \|r\|_2^2 + c^T x - r^T b + \tau \|A^T r - c\|_{\frac{1}{w}, \infty}$$

Dual when $\mu > 0$. The simplest way of dealing with $\mu > 0$ is to rewrite the problem as:

$$\underset{x}{\text{minimize}} \quad \frac{1}{2} \|\tilde{A}x - \tilde{b}\|_2^2 \quad \text{subject to} \quad \|x\|_{w,1} \leq \tau$$

with $\tilde{A} = [A; \sqrt{\mu}I]$, and $\tilde{b} = [b; 0]$. This reduces the problem to the form where $\mu = 0$ and we can therefore directly use dual formulation (28). Choosing $y = \tilde{r}$, with $\tilde{r} = [r; -\sqrt{\mu}x]$, and applying the same derivation as given above, we obtain a dual objective value of

$$r^T b - \tau \lambda - \frac{1}{2} \|r\|_2^2 - \frac{\mu}{2} \|x\|_2^2, \quad \text{with} \quad \lambda = \|A^T r - \mu x - c\|_{\frac{1}{w}, \infty}, \tag{29}$$

and a corresponding duality gap of

$$\delta = \|r\|_2^2 + c^T x - r^T b + \mu \|x\|_2^2 + \tau \|A^T r - \mu x - c\|_{\frac{1}{w}, \infty}. \tag{30}$$

Another approach is to solve the original infimum over x in (27) for the case where $\mu > 0$. For a fixed y and λ we have

$$\begin{aligned}
m(y, \lambda) &:= \inf_x \{ (c - A^T y)^T x + \frac{\mu}{2} \|x\|_2^2 + \lambda \|x\|_{w,1} \} \\
&= \mu \inf_x \left\{ -\frac{1}{\mu} (A^T y - c)^T x + \frac{1}{2} \|x\|_2^2 + \frac{\lambda}{\mu} \|x\|_{w,1} \right\} \tag{31}
\end{aligned}$$

When $\lambda = 0$ it is easily seen that $x^* = \frac{1}{\mu}(A^T y - c)$, thus giving $m(x) = -\frac{1}{2\mu}\|A^T y - c\|_2^2$. For the more general case where $\lambda > 0$, we first reformulate (31) as

$$m(y, \lambda) = \mu \inf_x \left\{ -v^T x + \frac{1}{2}\|x\|_2^2 + h(x) \right\}. \quad (32)$$

with $h(x) = \frac{\lambda}{\mu}\|x\|_{w,1} = \|x\|_{\frac{\lambda w}{\mu},1}$ and $v = \frac{1}{\mu}(A^T y - c)$. For problems of the form (32) we have:

Theorem 4.4. *Let $h(\cdot)$ be any norm then*

$$\inf_x -v^T x + \frac{1}{2}\|x\|_2^2 + h(x) = -\frac{1}{2}\|\text{prox}_h(v)\|_2^2$$

Proof. Note that the objective is coercive and therefore attains the minimum. This allows us to rewrite and solve the objective as follows:

$$u = \arg \min_x \frac{1}{2}\|x - v\|_2^2 + h(x) = \text{prox}_h(v).$$

We then need to show that

$$-v^T u + \frac{1}{2}\|u\|_2^2 + h(u) = -\frac{1}{2}\|u\|_2^2$$

From the Moreau decomposition we have $v = \text{prox}_h(v) + \text{prox}_{h^*}(v)$, where h^* is the conjugate of h . Using $\text{prox}_{h^*}(v) = v - u$ we have

$$\begin{aligned} -v^T u + \frac{1}{2}\|u\|_2^2 + h(u) &= -(u + (v - u))^T u + \frac{1}{2}u^T u + h(u) \\ &= -\frac{1}{2}\|u\|_2^2 - (v - u)^T u + h(u) \\ &= -\frac{1}{2}\|\text{prox}_h(v)\|_2^2 - \text{prox}_{h^*}(v)^T \text{prox}_h(v) + h(\text{prox}_h(v)) \\ &= -\frac{1}{2}\|\text{prox}_h(v)\|_2^2, \end{aligned}$$

where the last equality follows from Lemma 4.5 given below. \square

Lemma 4.5. *Let $h(\cdot)$ be any norm with conjugate $h^*(\cdot)$, then*

$$h(\text{prox}_h(x)) = \text{prox}_{h^*}(x)^T \text{prox}_h(x).$$

Proof. Let $u = \text{prox}_h(x)$, then it is well known that $\text{prox}_{h^*}(x) = x - u$ and $x - u \in \partial h(u)$. It thus remains to show that $h(u) = (x - u)^T u$. The subgradient $\partial h(u)$ and norm h are defined in terms of the dual norm h_* as

$$\partial h(u) := \arg \max_{w: h_*(w) \leq 1} w^T u, \quad \text{and} \quad h(u) = \max_{w: h_*(w) \leq 1} w^T u,$$

respectively, which means that $h(u) = w^T u$ for any $w \in \partial h(u)$. Choosing $w = x - u$ gives $h(u) = w^T u = (x - u)^T u$, as desired. \square

Application of Theorem 4.4 to (32) with proximal operator (see also (18))

$$\text{prox}_h(v) = \text{sign}(v) \left[|v| - \frac{\lambda w}{\mu} \right]_+,$$

we obtain

$$m(y, \lambda) = -\frac{\mu}{2} \left\| \text{sign}\left(\frac{1}{\mu}(A^T y - c)\right) \left[\left| \frac{1}{\mu}(A^T y - c) \right| - \frac{\lambda w}{\mu} \right]_+ \right\|_2^2 = -\frac{1}{2\mu} \left\| [A^T y - c] - \lambda w \right\|_2^2.$$

The same expression holds for $\lambda = 0$ and substitution into (27) therefore gives the following dual problem:

$$\underset{y, \lambda \geq 0}{\text{maximize}} \quad y^T b - \tau \lambda - \frac{1}{2} \|y\|_2^2 - \frac{1}{2\mu} \left\| [A^T y - c] - \lambda w \right\|_2^2. \quad (33)$$

Even when restricting y to the current residual r in the primal formulation, we can show that the value of (33) is never smaller than that of (29) and, consequently, that the duality gap never exceeds the value in (30). Choosing $\lambda = \|A^T y + \mu x - c\|_{\frac{1}{w}, \infty}$, means that for any index i we have

$$\lambda \geq \frac{1}{w_i} |[A^T x - c]_i + \mu x_i| \geq \frac{1}{w_i} |[A^T x - c]_i| - \frac{\mu}{w_i} |x_i|$$

Multiplying either side by w_i and rearranging gives

$$|[A^T y - c]_i| - \lambda w_i \leq \mu |x_i|.$$

Because the right-hand side is always nonnegative, this continues to hold when applying the $[\cdot]_+$ operator on the left-hand side, and as a result we have

$$\frac{1}{2\mu} \left\| [A^T y - c] - \lambda w \right\|_2^2 \leq \frac{\mu}{2} \|x\|_2^2,$$

from which the desired result immediately follows.

Finding a dual-feasible solution. It follows from Slater's condition and strong duality that, at the solution (x, r) for (28), we have $y = r$ and, without loss of generality, $\lambda = \|A^T r - c\|_{\frac{1}{w}, \infty}$. When r is not optimal, we can still choose $y = r$ and obtain a dual-feasible solution. For (33) we can also take $y = r$, but finding λ requires some more work. In general, given any y we want to find a λ that maximizes the objective. Writing $z = [A^T r - c]$ and ignoring constant terms, this is equivalent to solving

$$\lambda^* := \underset{\lambda \geq 0}{\text{argmin}} \quad \tau \lambda + \frac{1}{2\mu} \|[z - \lambda w]_+\|_2^2.$$

With $\mathcal{I}(\lambda) := \{i \mid z_i \geq \lambda w_i\}$ we can write the objective as

$$f(\lambda) = \tau \lambda + \frac{1}{2\mu} \sum_{i \in \mathcal{I}(\lambda)} (z_i - \lambda w_i)^2$$

Discarding all zero terms with $z_i = 0$, this function is piecewise quadratic with breakpoints at $\lambda_i = w_i/z_i$. We can write the sequence of breakpoints in non-decreasing order as $\lambda_{[i]}$ for $i = 0, \dots, n$, with $\lambda_{[0]} := 0$. The gradient between successive breakpoints is linear and continuously increases from $f'(0) = \tau - 1/\mu \sum_i w_i z_i$ to $f'(\lambda_{[n]}) = \tau$. In order to find the optimal point λ^* , we consider two cases. In the first case we have $f'(0) \geq 0$, or equivalently $\tau \geq 1/\mu \sum_i w_i z_i$, which means that the function is non-decreasing and we find $\lambda^* = 0$. In the second case we need to find λ for which the gradient vanishes. This can be done by traversing the breakpoints until the first breakpoint is found where the gradient is nonnegative. The desired solution λ^* is then found by linear interpolation over the last segment. Including sorting this can be done in $\mathcal{O}(n \log n)$ time. This problem is very similar to projection onto the one-norm ball, and can also be evaluated in expected $\mathcal{O}(n)$ time using an algorithm similar to that proposed in [13].

Primal-dual pairs. Using the methods described above, we can compute an upper bound on the duality gap given a feasible x and the corresponding residual r . We can do at least as good, and often better, by maintaining the maximum dual objective found so far, and using this to determine the relative duality gap. This way it is possible for the primal objective for the current iterate x to attain the desired optimality tolerance while the corresponding dual estimate is far from optimal. Within the root-finding framework this means that we cannot simply use the latest residual r to evaluate the gradient of the Pareto curve. Instead we should maintain the value of λ corresponding to the best dual solution at any point, and use this as the gradient approximation. In other words, we need to keep track of the best primal and dual variables separately.

5 Numerical Experiments

In this section we evaluate the performance of the hybrid approach on the Lasso problem (LS_τ) both independently and within the SPGL1 root-finding framework [2] described in the introduction. The SPGL1 solver can be used both for stand-alone Lasso problems, as well as for basis-pursuit denoise (BP_σ) problems. For the hybrid method we are mostly concerned with the performance and the former and we therefore changed SPGL1 in two stages. First we modified the stopping criteria used in the Lasso mode, now declaring a solve successful only if the relative duality falls below a certain tolerance level. We then added all modifications needed for the implementation of the hybrid approach. To distinguish between the different algorithms, we use the convention that SPGL1 is used only to refer to the existing implementation provided by [2]. We refer to the version of SPGL1 with the more stringent stopping criteria as the SPG method, which is then extended with the techniques described in this paper to obtain the hybrid method.

When used in the root-finding mode to solve (BP_σ), SPGL1 uses several different criteria to decide when to update τ . Each subproblems in SPGL1 is considered solved when the relative change in objective is small, and at least one iteration was taken within the current subproblem. The overall problem is declared solved when $\|A^T y\|_\infty$, the relative difference between $\|r\|_2$ and σ , or the relative duality gap is sufficiently small. For the basis-pursuit denoise experiments based on the SPG and hybrid algorithms, we use a separate implementation of the root-finding framework in which each Lasso subproblem is fully solved before updating τ . The differences in stopping criteria, and especially the lack of guarantees on the duality gap for the final subproblem in SPGL1, make it difficult to compare the performances directly. We therefore focus predominantly on how the performance of the hybrid method differs from the reference SPG method.

5.1 Lasso on sparse problems

In the first set of experiments we compare the performance of backtracking line search and line search along the entire projection trajectory for both the SPG and hybrid method. For the trajectory line search we return either with the first local minimum or with the global minimum. For the test problems we follow a conventional compressed-sensing scenario where A is a random 1024×2048 matrix with i.i.d. normal entries with columns normalized to unit norm. We set $b = Ax_0$, for k -sparse vectors x_0 with random support and entries generated i.i.d. from (1) the normal distribution; (2) the uniform distribution over $[-1, 1]$; and (3) the discrete set $\{-1, +1\}$ with equal probability. We set $\tau = 0.995 \cdot \|x_0\|_1$ and terminate the algorithm whenever the relative duality gap, computed as $(f(x) - f_{\text{dual}}) / \max\{f(x), 10^{-3}\}$, falls below a given tolerance level.

Table 1 shows the average runtime and number of iterations for twenty random instances of

k	Runtime (sec.)						Iterations					
	Backtrack		Local trajectory		Global trajectory		Backtrack		Local trajectory		Global trajectory	
50	0.14	0.17	0.46	0.40	0.53	0.46	25	29	22	24	22	24
	0.13	0.17	0.44	0.39	0.52	0.46	24	28	20	22	20	23
	0.14	0.18	0.45	0.37	0.50	0.46	25	30	21	22	21	22
100	0.22	0.26	0.69	0.58	0.93	0.82	40	44	35	34	34	34
	0.22	0.25	0.68	0.57	0.92	0.82	40	43	34	33	33	33
	0.23	0.26	0.69	0.57	0.96	0.83	41	45	34	34	34	34
150	0.34	0.38	0.97	0.83	1.56	1.39	59	62	54	54	53	53
	0.33	0.36	0.96	0.80	1.50	1.41	57	59	51	51	50	51
	0.37	0.40	0.94	0.80	1.57	1.39	62	64	53	53	50	51
200	0.55	0.57	1.52	1.32	2.81	2.49	88	88	93	92	89	88
	0.53	0.55	1.50	1.29	2.78	2.55	86	87	95	93	88	87
	0.59	0.61	1.52	1.31	2.88	2.64	95	94	100	98	87	88

(a) Tolerance 10^{-4}

50	0.16	0.19	0.52	0.42	0.59	0.48	28	32	25	26	25	26
	0.16	0.19	0.51	0.41	0.58	0.48	28	31	24	25	24	26
	0.24	0.20	0.53	0.39	0.57	0.45	32	33	25	25	25	25
100	0.34	0.28	0.75	0.60	1.01	0.84	48	47	38	38	37	37
	0.25	0.28	0.72	0.59	0.97	0.84	44	47	36	37	36	37
	0.31	0.29	0.76	0.62	1.05	0.88	48	48	37	38	37	38
150	0.50	0.42	1.08	0.89	1.69	1.46	70	67	59	59	57	58
	0.45	0.39	1.00	0.85	1.57	1.38	67	64	55	56	54	55
	0.60	0.44	1.04	0.82	1.61	1.41	75	69	58	58	54	56
200	0.84	0.60	1.70	1.34	2.92	2.52	107	95	102	98	97	94
	0.80	0.61	1.65	1.33	2.96	2.58	104	94	102	100	96	94
	0.88	0.65	1.70	1.37	3.06	2.66	112	101	107	105	95	95

(b) Tolerance 10^{-6}

Table 1: *Comparison of different line-search methods. From top to bottom for each sparsity level are the results for random Gaussian, uniform, and discrete sign support. The left and right columns correspond to the SPG and the hybrid methods, respectively. Runtimes and iterations are averaged over twenty problem instances.*

each test problem for different sparsity levels k , and each of the three distributions used for the on-support values in x_0 . Comparing across the line search methods we see that the two trajectory line search methods require fewer iterations than backtracking to converge. The backtracking line search, on the other hand, has a lower per-iteration cost and overall outperforms the trajectory line search uniformly in terms of runtime. Looking at the difference between the SPG and hybrid methods, we see that the number of iterations required by the hybrid method is larger than that of the SPG method for the lower optimality tolerance and smaller values of k . Combined with the backtracking line search this means that the runtime of the hybrid method is slightly larger compared to the SPG method. For the trajectory line search we see, perhaps somewhat surprisingly, that the runtime of the hybrid method is uniformly lower than the SPG method, despite the larger number of iterations. The reason for this is that the line search for each quasi-Newton iteration taken by the hybrid method is much faster than the trajectory line search, thereby reducing the overall runtime. Comparing between the two optimality tolerance levels we note that the hybrid method does well for the lower tolerance level of 10^{-6} . The SPG method is

k	Runtime SPG (s)			rel.gap	✓	Runtime hybrid (s)			rel.gap	✓	(%)
50	0.17	0.16	0.16	3.8e-7	100	0.20	0.19	0.19	4.0e-7	100	-17
75	0.24	0.21	0.21	5.8e-7	99	0.24	0.23	0.23	4.7e-7	100	-7
100	0.28	0.27	0.31	5.8e-7	95	0.28	0.27	0.28	5.3e-7	100	3
125	0.39	0.36	0.37	6.5e-7	91	0.34	0.32	0.31	6.2e-7	100	14
150	0.44	0.42	0.45	7.3e-7	92	0.39	0.39	0.37	6.7e-7	100	13
175	0.64	0.53	0.55	7.7e-7	81	0.46	0.44	0.45	6.9e-7	100	21
200	0.74	0.66	0.73	8.2e-7	75	0.58	0.54	0.53	7.1e-7	100	23
225	0.90	0.88	0.86	8.5e-7	71	0.71	0.68	0.65	7.5e-7	100	23
250	1.30	1.10	1.17	9.7e-7	55	0.93	0.86	0.84	7.6e-7	100	26
275	1.65	1.54	1.42	1.0e-6	50	1.30	1.14	1.07	8.3e-7	100	24
300	2.39	2.01	1.85	2.1e-6	35	1.86	1.53	1.44	8.3e-7	100	23
325	3.60	2.76	2.66	4.7e-6	26	3.08	2.04	2.00	8.8e-7	100	22
350	6.61	4.31	3.91	7.6e-6	15	5.68	3.30	3.08	9.0e-7	100	19
375	31.82	8.27	7.15	1.5e-5	7	20.09	6.19	5.68	9.1e-7	100	28
400	218.46	23.16	16.16	3.0e-5	9	129.42	14.65	10.93	9.5e-7	89	37

Table 2: Comparison between the SPG and the hybrid method on exact sparse problems with k non-zero entries. The first three columns for either method give the average runtime over 50 instances when the non-zero entries are sampled i.i.d. from respectively the discrete $\{-1, 1\}$, uniform $(-1, 1)$, and the normal distribution. The fourth column gives the median relative duality gap at the final iteration taken over all 150 problem instances and should be compared with the optimality tolerance, which was set to 10^{-6} . The fifth column for each of the two blocks, indicated by the check mark, gives the percentage of runs that completed successfully, that is, completed without a line-search error. The right-most column gives the average of the speed up values for each of the three distributions.

not only slower for these problems, but also suffers from a problem where the line search fails to find a feasible step length before the desired optimality tolerance is reached, thereby terminating the optimization prematurely. This problem did not occur for a tolerance level of 10^{-4} , but for 10^{-6} this happened on one of the twenty problems for $k = 50$ and gradually went up to five out of twenty for $k = 200$. No such line-search errors occurred in the hybrid method.

From the results in Table 1, along with various other experiments not shown here, it was found that backtracking line search outperforms the trajectory line search. As a result, we only consider the former throughout the remainder of this section. We now take a closer look at the occurrence of line-search errors and the speed up obtained using the hybrid method. For this, we modify the earlier setup by increasing the range of sparsity levels k and choosing $\tau = 0.99 \cdot \|x\|_1$. We run 50 instances for each of the three distributions used above and report in Table 2 the results obtained with an optimality tolerance of 10^{-6} . In general we see that the runtime goes up considerable as we keep increasing k . Moreover, the results show clear differences in the runtime for the three distributions with a much higher runtime for problems based on sparse vectors with ± 1 entries. For sparsity levels up to around one hundred the number of iterations in the SPG method is relatively small (between 25 and 50). For these problems the hybrid method may complete before or soon after the first quasi-Newton step is taken. The slight overhead of the method and occasionally a small number of additional iterations make the hybrid method somewhat slower on average for these problems than the SPG method. For larger values of k , the number of iterations goes up, and the effect of the quasi-Newton steps in the hybrid method becomes apparent with average speed up values between 20 and 30%. Aside from reduced runtime we see from Table 2 that the hybrid method also manages to solve problems to the desired accuracy level much better than the SPG method. The number of solved problems steadily falls to around 9% with increasing k for the SPG method, but remains at 100% for all but the largest k for the hybrid method. The

median relative duality gap provides further information about the level of accuracy reached before the algorithm completes or terminates with a line-search error. For the largest values of k , the SPG method fails to complete with a relative duality gap of even 10^{-5} for at least half of the problems.

5.2 Root finding

Given that most of the runtimes that appear in Tables 1 and 2 are of the order of seconds, it is valid to question whether these problems are too idealized and well behaved to give a good idea about the practical performance of the algorithms. In this section we therefore look at two different types of problems. First we introduce a class of random problems that better reflect conditions found in practical problems. Second we evaluate the performance on the Sparco [4] collection of test problems for sparse reconstruction.

5.2.1 Coherent test problem generation

In the compressed-sensing literature it is well known that a random Gaussian matrix satisfies with high probability that all sufficiently small subsets of columns form a near-orthogonal basis for the subspace spanned by these columns—a property known as the restricted isometry [10]. Another quantity used to characterize matrices is the mutual coherence, defined as the maximum absolute pairwise cosine distance between the columns. In practical applications matrix A is often more coherent [8]. Although there are no theoretical results on how this affects the complexity of one-norm minimization, it has been observed empirically that more coherent problems are harder to solve. The construction we propose for generating such problems is by means of a random walk on the $(m - 1)$ -sphere with a step size parameterized by γ . Starting with a unit norm column a_1 we construct successive columns by sampling a vector v_k with i.i.d. Gaussian entries and setting $a_{k+1} = \alpha_1 a_k + \alpha_2 v_k$, where α_1 and α_2 are chosen such that $\|a_{k+1}\|_2 = 1$ and $\langle a_k, a_{k+1} \rangle = 1 - \gamma$. In other words, a_{k+1} lies on the boundary of a spherical cap with center a_k and angle θ such that $\cos(\theta) = 1 - \gamma$. The mutual coherence of the resulting matrix is lower bounded by $1 - \gamma$, and an example of the distribution of the pairwise cosine distance between the columns is given in Figure 5(a). An example Gram matrix, plotted in Figure 5(b), shows that aside from the banded structure, there are regions of increased coherence whenever the random walk approaches earlier locations. From Figure 5(c) we see that lowering γ while keeping a_1 and v_k fixed leads to an increase of the top singular value σ_1 as the columns become more and more similar. Figure 5(d) illustrates that the maximum pairwise coherence μ does not necessarily have a relationship with the top singular value.

5.2.2 Highly coherent measurement matrices

We apply the SPG and hybrid method to solve (BP_σ) using the root-finding framework explained in Section 1. Each Lasso subproblem (LS_τ) is optimized to a certain optimality tolerance, and the overall problem is considered solved whenever the relative misfit $|\sigma - \|r\|_2| / \max(\sigma, 10^{-3})$ falls below 10^{-5} . For completeness we also compare the performance with the SPGL1 algorithm as provided by [2].

For the first set of experiments we use the highly coherent matrices described in Section 5.2.1. As before we create a k -sparse vector x_0 with non-zero entries sampled from different distributions, and set $b = Ax_0 + v$, where the entries in v are zero in the noiseless case, and sampled i.i.d. from the normal distribution and scaled to the desired noise level otherwise. For the noiseless results

γ	SPGL1 runtime	Root finding tolerance 10^{-4}			Root finding tolerance 10^{-6}			Lasso tolerance 10^{-4}			Lasso tolerance 10^{-6}		
.100	0.7	0.7	0.8	-4	0.7	0.8	-4	0.7	0.7	-4	0.7	0.7	-3
.050	1.6	1.7	1.7	2	1.9	1.7	12	1.8	1.7	1	1.9	1.8	7
.020	6.8	6.6	5.8	12	7.6	5.9	22	6.5	6.0	7	7.1	6.1	14
.010	17.8	17.7	14.7	17	21.4	15.0	30	16.7	15.5	7	18.3	15.6	15
.005	51.9	43.6	36.8	16	52.6	38.2	27	40.6	37.9	7	42.7	38.2	11
(a) Sparsity $k = 10$													
.100	1.8	3.1	2.7	11	5.0	3.2	37	1.8	1.7	7	2.5	1.8	29
.050	5.1	8.2	7.1	13	11.3	7.9	30	4.7	4.3	8	5.8	4.6	21
.020	19.8	29.4	23.2	21	36.5	25.8	29	17.5	15.5	11	18.9	16.1	15
.010	48.5	77.7	58.9	24	92.5	63.0	32	45.9	40.0	13	47.9	41.2	14
.005	145.7	221.9	144.9	35	263.4	153.4	42	127.7	101.9	20	134.9	104.7	22
(b) Sparsity $k = 50$													
.100	3.2	5.2	4.5	13	7.8	5.3	32	2.8	2.6	8	3.6	2.8	23
.050	8.6	14.9	12.4	17	19.4	13.8	29	7.6	6.9	10	8.8	7.3	17
.020	29.9	55.1	41.7	24	65.8	45.2	31	29.3	25.3	14	31.0	26.1	16
.010	72.4	158.9	110.5	30	188.0	115.3	39	84.4	68.7	19	88.3	70.5	20
.005	224.2	502.7	302.4	40	596.0	312.6	48	251.3	187.1	26	261.5	189.8	27
(c) Sparsity $k = 100$													
.100	2.2	4.7	4.2	12	8.3	5.3	37	2.3	2.1	9	2.9	2.3	22
.050	7.3	12.5	10.0	20	18.1	12.0	34	5.8	5.4	8	6.7	5.7	15
.020	28.7	43.6	33.1	24	54.9	37.7	31	23.0	19.9	14	24.4	20.6	15
.010	68.8	118.0	84.1	29	140.5	93.4	34	59.4	51.5	13	62.3	52.9	15
.005	233.1	367.5	221.5	40	447.5	234.4	48	180.1	140.6	22	189.9	143.0	25
(d) Sparsity $k = 50$, noise level 1%													
.100	1.8	3.9	3.3	13	5.6	4.2	26	1.6	1.5	4	2.0	1.6	16
.050	4.6	9.3	7.7	17	13.1	9.3	29	3.8	3.6	4	4.5	3.9	13
.020	15.2	30.8	23.7	23	42.3	27.1	36	13.0	11.4	12	15.1	12.1	20
.010	32.4	78.5	55.2	30	109.6	62.3	43	33.2	27.9	16	39.2	28.7	27
.005	66.5	219.2	131.4	40	334.7	139.7	58	90.2	68.6	24	115.8	70.1	39
(e) Sparsity $k = 50$, noise level 5%													
.100	1.5	3.4	3.0	10	4.5	3.7	18	1.4	1.3	8	1.7	1.4	13
.050	3.3	7.9	6.7	16	10.9	8.2	25	3.2	2.8	13	3.8	3.1	18
.020	9.5	24.7	19.1	23	34.3	22.2	35	10.0	8.3	17	12.0	8.8	27
.010	19.5	61.7	44.6	28	85.9	50.9	41	24.5	19.2	22	29.1	20.2	31
.005	33.2	154.5	98.1	37	263.1	107.7	59	60.5	43.1	29	84.7	44.8	47
(f) Sparsity $k = 50$, noise level 10%													

Table 3: Comparison between SPGL1 and root finding with strict tolerance levels using the SPG and hybrid method. The columns within the root finding and Lasso blocks are respectively the runtime in seconds of the SPG and hybrid method, and (in blue) the reduction in runtime in percent of the hybrid method compared to the SPG method.

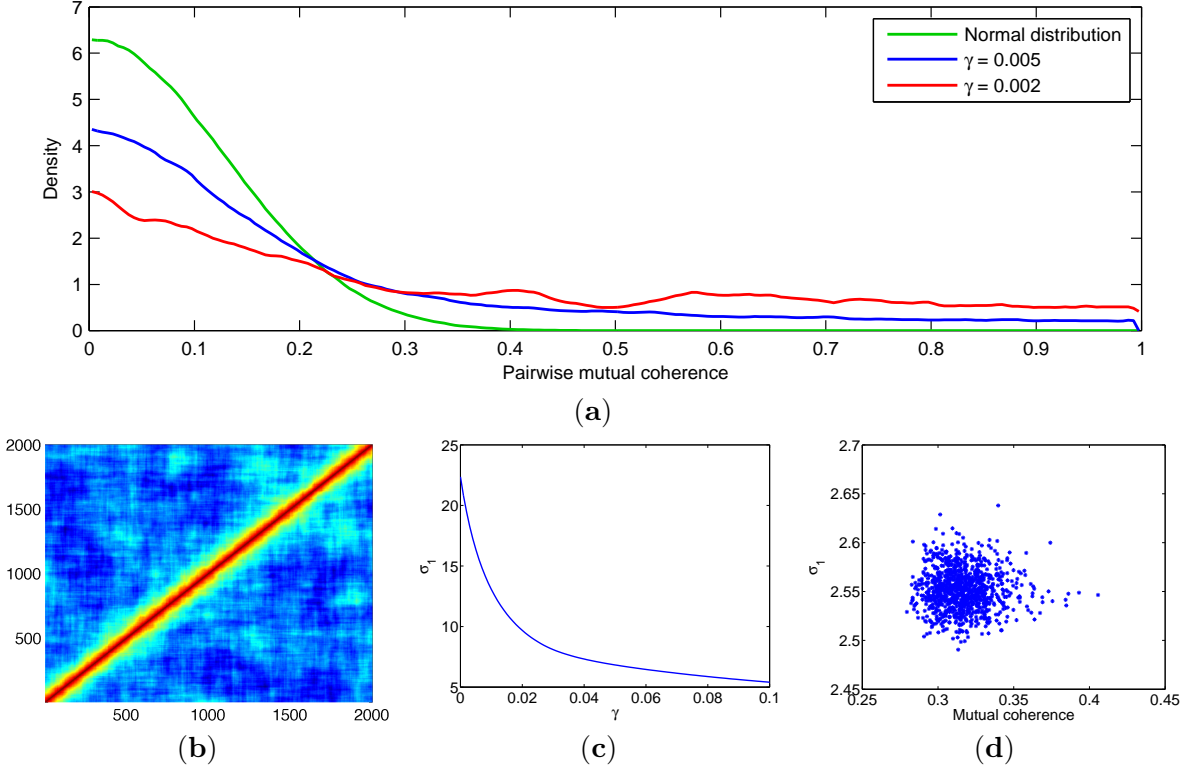


Figure 5: (a) Distribution of pairwise mutual coherence between vectors of two types of 64×2048 matrices with unit-normalized columns generated as: (i) random vectors with i.i.d. normal entries, and (ii) a random walk over the sphere with the mutual coherence between successive columns equal to $1 - \gamma$; (b) Gram matrix of a 200×2000 matrix generated with $\gamma = 0.01$; (c) the top singular value of a 200×500 matrix generated with the same a_1 and v_k for different values of λ ; and (d) the mutual coherence and top singular value for 1000 random Gaussian 200×500 matrices with columns scaled to unit norm.

in Tables 3(a)–(c) we run ten instances for each of the three distributions and report the average run time over all thirty runs. The percentage time reduction is computed based on the total runtime and matches the percentage obtained for each of the three signal classes independently. For the root-finding columns we solve (BP_σ) with $\sigma = 0.01\|b\|_2$ and optimality tolerance levels of 10^{-4} and 10^{-6} . For the Lasso columns we solve (LS_τ) on equivalent problems with τ set to the value obtained using the root-finding procedure. The results in Tables 3(d)–(f) apply to noisy problems where $\|v\|_2$ is scaled to the given percentage of $\|Ax_0\|_2$, and σ is set accordingly. For these experiments we only consider sparse x_0 with random ± 1 entries. Table 4 summarizes the total runtime for the different solvers along with percentage of solutions that have a relative duality gap within the given ranges.

The first thing to note from the results in Table 3 is that the problems generated with lower γ values are indeed more difficult to solve for both the SPG and the hybrid method. Compared to the SPG method, the hybrid method reduces the average runtime for nearly all problems, and does so by a percentage that increases as the problems get harder. From Table 4 we see that the hybrid method with optimality tolerances of 10^{-4} and 10^{-6} reduces the total runtime respectively by 34% and 43% for the basis-pursuit problems, and 20% and 24% for the Lasso problems. The larger relative reduction in runtime for basis pursuit is due to the use of warm starting in the root-finding procedure, which removes a substantial number of iterations that would otherwise be identical for the hybrid and SPG methods. Despite the improvements, the hybrid method still

Type	Method	Tol.	Time	Relative duality gap					
				$\leq 10^{-6}$	$10^{(-6,-5]}$	$10^{(-5,-4]}$	$10^{(-4,-3]}$	$10^{(-3,-2]}$	$> 10^{-2}$
BP_σ	SPGL1	10^{-6}	7h25	0.2	1.5	4.6	15	49	29
	SPG	10^{-6}	17h29	39	33	28	0.2	—	—
	hybrid	10^{-6}	9h53	100	—	—	—	—	—
	SPG	10^{-4}	14h02	0.2	0.6	99	0.5	—	—
	hybrid	10^{-4}	9h18	0.2	0.8	99	—	—	—
LS_τ	SPG	10^{-6}	7h55	36	33	30	0.5	—	—
	hybrid	10^{-6}	6h02	100	—	—	—	—	—
	SPG	10^{-4}	7h20	—	1.2	98	0.5	—	—
	hybrid	10^{-4}	5h54	—	1.2	99	—	—	—

Table 4: *Total runtime for the coherent problems with different methods and optimality tolerances, along with the percentage of instances that attain a relative duality gap in the given intervals at the final iteration. The reduction in runtime for the successive SPG-hybrid pairs are 43, 34, 24, and 20%, respectively.*

has a larger runtime than SPGL1 on most problems. However, from Table 4 we see that SPGL1 does not even reach a relative duality gap of 10^{-3} for nearly 80% of the problems, as a result of the relaxed stopping criteria. Tightening these criteria, as done in what we label the SPG method, increases the number of solutions that attain the desired optimality tolerance. Nevertheless, the SPG method still fails to reach an optimality of 10^{-6} for some 60% of the problems. Finally, we see that the hybrid method not only improves the runtime of the SPG method, but also manages to reach the requested optimality on all problems from Table 3.

5.2.3 Sparco test problems

Sparco [4] provides a standard collection of test problems for compressed sensing and sparse recovery. The problems in Sparco are of the form $b = Ax + v$, where A is represented as a linear operator rather than an explicit matrix. After excluding problems that are too easy to solve or require access to third-party software, we obtain the problem selection listed in Table 5. For some problems we scale the original b to avoid a very small objective value at the solution, which causes the duality gap relative to $\max(f(x), 1)$ to be satisfied more easily. The table also lists the one-norm of the solutions found when solving with $\sigma = 0.01\|b\|_2$ and $\sigma = 0.001\|b\|_2$, respectively, for the scaled b .

We run the SPG and hybrid methods with optimality tolerances ranging from 10^{-1} down to 10^{-4} . Beyond that, some of the problems simply took too long to finish. For SPGL1 we use optimality tolerance values set to 10^{-6} and 10^{-9} . By comparison these may seem excessively small, and we certainly do not expect the relative duality gap to reach these levels. Instead, we choose the small values to help control the other stopping criteria, such as the relative change in the objective value, which are parameterized using the same tolerance parameter. The results of the experiments with the two choices of σ , are summarized in Tables 6 and 7. The hybrid method reduces the runtime of the SPG method in 42 out of the 56 settings, often considerably so. For a tolerance level of 10^{-4} the hybrid method consistently outperforms the SPG method with an average time reduction of 38%. The required optimality level is reached on all problems except for problem 903 with the smaller σ and optimality tolerance 10^{-4} . For this problem the SPG method stops with a relative duality gap of $2 \cdot 10^{-4}$ following a line-search error. The runtime for SPGL1 with optimality tolerance 10^{-6} is very low overall, but comes at the cost of a rather large relative duality gap at the solution. Lowering the tolerance to 10^{-9} reduces the gap, but also leads to a considerable increase in runtime. In either case the number of root-finding iterations can be very

Problem	ID	m	n	$\ b\ _2$	scale	$\ x_{1e-2}^*\ _1$	$\ x_{1e-3}^*\ _1$
blurrycam	701	65536	65536	1.3e+2	100	2.8e+5	9.1e+5
blurspike	702	16384	16384	2.2e+0	100	2.5e+4	3.4e+4
soccer1	601	3200	4096	5.5e+4	1	7.9e+1	3.1e+2
spiketrn	903	1024	1024	5.7e+1	100	1.3e+3	1.3e+3
yinyang	603	1024	4096	2.5e+1	100	2.5e+4	2.6e+4
srcsep1	401	29166	57344	2.2e+1	1	1.0e+3	1.0e+3
srcsep2	402	29166	86016	2.3e+1	1	1.1e+3	1.1e+3

Table 5: *Selected sparco problems.*

large, especially if the target value of τ is exceeded and gradual reduction follows. The lowest relative duality gap reached by SPGL1 over all problems in Tables 6 and 7 is $4 \cdot 10^{-3}$. The varying optimality levels make it difficult to compare results, so of special interest are problem instances where SPGL1 simultaneously has a lower runtime and relative duality gap with either the SPG or hybrid method, or vice versa. From the tables we see that SPGL1 outperforms the SPG method on both instances of problem 702. For problem 401 in Table 6, SPGL1 with an optimality tolerance of 10^{-6} is better, but aside from this problem, SPGL1 consistently has the lowest runtime, but also the largest duality gap. The SPG method with more stringent root-finding iterations dominates SPGL1 with a tolerance level of 10^{-9} on all remaining problems aside from the instance of problem 701 in Table 6. As we saw earlier, the hybrid method performs especially well when the desired relative duality gap is small. Nevertheless, even for the large duality gaps in question it still dominates SPGL1 on nine out of the fourteen problem instances and is dominated on only one.

5.2.4 Primal-dual gap

We now consider the formulation

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|Ax - b\|_2^2 + \frac{\mu}{2}\|x\|_2^2 \quad \text{subject to} \quad x \in \mathcal{C}, \quad (34)$$

for $\mu > 0$. In Section 4.3 we described two different ways of deriving a dual formulation. In the first approach we augment A and b to account for the $\frac{\mu}{2}\|x\|_2^2$ term and reduce the problem to the standard Lasso formulation. The derivation of the dual for this formulation in [2, 3] provides a way of generating a dual-feasible point $(\bar{y}, \bar{\lambda})$ from a primal-feasible x by choosing $\bar{y} = \bar{A}x - \bar{b}$ and solving a trivial optimization problem for $\bar{\lambda}$. In the second approach we deal with formulation (34) directly and obtain a dual problem parameterized in (y, λ) . As before we can choose y to be equal to the residual, now in terms of the original A and b , and remain with a non-trivial optimization problem for λ that is nevertheless easily solved using the algorithm described in Section 4.3. We refer to the two derivations as the augmented derivation and the optimized derivation. The term ‘optimized’ refers to the need to solve for λ , but more importantly, to the fact that the dual objective generated from any x using the optimized derivation is never smaller than that using the augmented derivation, as shown in Section 4.3.

To evaluate the practical difference between the two approaches we generate a large number of randomized test problems of the form $b = Ax_0 + v$, where x_0 are random vectors with sparsity levels ranging from 50 to 350 in steps of 50 and on-support entries draw i.i.d. from the normal distribution. The $m \times n$ measurement matrices A are drawn i.i.d. from $\mathcal{N}(0, 1/m)$, with $m = 1000$ and $n = 2000$. Finally, the additive noise vectors v are drawn from the normal distribution and scaled to have Euclidean norm equal to 0, 0.01, 0.1, and 1 percent of that of the clean observation Ax_0 . For the problem formulation we set τ to $\|x_0\|_1$ scaled by 0.7 through 1.2 in 0.1 increments,

	SPGL1		SPG		SPG		SPG		SPG	
	10 ⁻⁶	10 ⁻⁹	hybr. Tol = 10 ⁻¹		hybr. Tol = 10 ⁻²		hybr. Tol = 10 ⁻³		hybr. Tol = 10 ⁻⁴	
Runtime (s)	18.0	32.2	24.3	24.9	44.6	54.0	84.1	75.9	91.1	66.7
Rel.gap	3e-1	2e-2	9e-2	9e-2	9e-3	9e-3	9e-4	1e-3	1e-4	8e-5
Outer iterations	144	246	12	13	12	12	12	10	10	10
► Problem 701 – blurrycam				-2.6		-21.2		9.7		26.8
Runtime (s)	9.7	16.7	17.5	18.2	27.8	28.0	38.9	38.2	38.2	35.4
Rel.gap	6e-1	3e-2	9e-2	1e-1	9e-3	1e-2	1e-3	1e-3	1e-4	1e-4
Outer iterations	191	296	11	11	10	10	10	10	8	8
► Problem 702 – blurspike				-3.9		-0.5		1.8		7.2
Runtime (s)	133	206	24.6	11.5	28.0	20.1	34.0	17.7	38.0	17.2
Rel.gap	1.5	6e-2	5e-2	5e-2	7e-3	6e-3	1e-3	4e-4	1e-5	2e-5
Outer iterations	1274	1945	13	11	9	9	12	9	8	9
► Problem 601 – soccer1				53.2		28.2		47.8		54.7
Runtime (s)	2.8	6.6	3.2	2.9	5.6	3.8	7.6	5.6	11.2	5.3
Rel.gap	3.6	1e-1	9e-2	1e-1	8e-3	8e-3	1e-3	8e-4	1e-4	1e-4
Outer iterations	448	1120	17	8	6	6	6	10	5	5
► Problem 903 – spiketrn				9.0		31.3		27.2		53.0
Runtime (s)	1.8	2.7	2.1	2.1	2.5	2.9	2.9	3.1	4.2	4.0
Rel.gap	3e-1	2e-2	8e-2	9e-2	7e-3	6e-3	8e-4	8e-4	7e-5	8e-5
Outer iterations	38	60	10	11	7	8	7	7	7	7
► Problem 603 – yinyang				-2.1		-13.5		-4.5		3.7
Runtime (s)	79.0	418	88.1	75.6	299	331	1404	1047	6351	2447
Rel.gap	5e-2	1e-1	7e-2	8e-2	9e-3	9e-3	1e-3	1e-3	1e-4	1e-4
Outer iterations	37	213	10	16	9	20	8	9	9	8
► Problem 401 – srcsep1				14.2		-10.5		25.5		61.5
Runtime (s)	114	622	151	146	441	409	1114	1023	2447	1565
Rel.gap	5e-1	1e-1	8e-2	7e-2	1e-2	9e-3	9e-4	1e-3	1e-4	6e-5
Outer iterations	30	198	10	11	9	9	8	7	7	9
► Problem 402 – srcsep2				3.3		7.1		8.2		36.1

Table 6: *Sparco* problems with $\sigma = 0.01\|b\|_2$, (a) runtime in seconds, (b) relative duality gap, (c) number of root-finding iterations. The percentage reduction in time of the hybrid method over SPG is given in blue next to the problem index.

	SPGL1		SPG		hybr.		SPG		hybr.		SPG		hybr.		SPG		hybr.	
	10 ⁻⁶	10 ⁻⁹	Tol = 10 ⁻¹		Tol = 10 ⁻²		Tol = 10 ⁻³		Tol = 10 ⁻⁴		Tol = 10 ⁻³		Tol = 10 ⁻⁴		Tol = 10 ⁻³		Tol = 10 ⁻⁴	
Runtime (s)	121	200	184	195	272	300	316	342	430	333	316	342	430	333	316	342	430	333
Rel.gap	2.1	1e-1	7e-2	7e-2	9e-3	6e-3	1e-3	8e-4	9e-5	9e-5	1e-3	8e-4	9e-5	9e-5	1e-3	8e-4	9e-5	9e-5
Outer iterations	485	640	13	13	12	12	11	12	11	11	11	12	11	11	11	12	11	11
► Problem 701 – blurrycam				-6.4		-10.4		-8.2		22.5								
Runtime (s)	28.8	34.9	35.4	32.7	45.8	49.3	56.1	59.8	74.4	68.1	56.1	59.8	74.4	68.1	56.1	59.8	74.4	68.1
Rel.gap	4e-1	7e-2	9e-2	9e-2	9e-3	1e-2	9e-4	9e-4	9e-5	9e-5	9e-4	9e-4	9e-5	9e-5	9e-4	9e-4	9e-5	9e-5
Outer iterations	127	151	13	13	11	11	10	10	10	10	10	10	10	10	10	10	10	10
► Problem 702 – blurspike				7.5		-7.7		-6.5		8.5								
Runtime (s)	278	368	84.3	47.6	155	56.4	203	53.2	257	47.8	203	53.2	257	47.8	203	53.2	257	47.8
Rel.gap	1.1	7e-2	8e-2	7e-2	9e-3	8e-3	1e-3	7e-4	1e-4	7e-5	1e-3	7e-4	1e-4	7e-5	1e-3	7e-4	1e-4	7e-5
Outer iterations	2059	2707	19	10	11	10	9	9	9	10	9	9	9	10	9	9	9	10
► Problem 601 – soccer1				43.5		63.6		73.8		81.4								
Runtime (s)	4.8	9.4	6.4	5.0	9.2	6.2	11.7	6.6	13.5	6.6	11.7	6.6	13.5	6.6	11.7	6.6	13.5	6.6
Rel.gap	22.5	1.3	5e-2	9e-2	1e-2	1e-2	1e-3	9e-4	2e-4	1e-4	1e-3	9e-4	2e-4	1e-4	1e-3	9e-4	2e-4	1e-4
Outer iterations	700	1253	6	6	8	8	6	8	5	5	6	8	5	5	6	8	5	5
► Problem 903 – spiketrn				21.9		33.1		43.7		51.0								
Runtime (s)	9.0	19.0	17.8	21.7	37.4	26.8	55.1	29.4	71.8	35.8	55.1	29.4	71.8	35.8	55.1	29.4	71.8	35.8
Rel.gap	1.2	9e-1	1e-1	8e-2	1e-2	7e-3	9e-4	1e-3	1e-4	1e-4	9e-4	1e-3	1e-4	1e-4	9e-4	1e-3	1e-4	1e-4
Outer iterations	90	190	16	15	8	10	8	8	8	8	8	8	8	8	8	8	8	8
► Problem 603 – yinyang				-21.4		28.4		46.7		50.1								
Runtime (s)	241	3309	108	85.0	394	360	2803	2253	21829	11193	2803	2253	21829	11193	2803	2253	21829	11193
Rel.gap	3e-3	4e-3	8e-2	6e-2	1e-2	9e-3	1e-3	1e-3	1e-4	1e-4	1e-3	1e-3	1e-4	1e-4	1e-3	1e-3	1e-4	1e-4
Outer iterations	32	264	46	21	13	12	20	11	10	10	20	11	10	10	20	11	10	10
► Problem 401 – srcsep1				21.6		8.5		19.6		48.7								
Runtime (s)	272	5999	191	178	518	477	2041	1787	8631	6486	2041	1787	8631	6486	2041	1787	8631	6486
Rel.gap	6e-3	2e-2	6e-2	8e-2	8e-3	8e-3	1e-3	9e-4	1e-4	9e-5	1e-3	9e-4	1e-4	9e-5	1e-3	9e-4	1e-4	9e-5
Outer iterations	18	324	240	192	11	14	9	10	10	10	9	10	10	10	9	10	10	10
► Problem 402 – srcsep2				6.5		8.0		12.5		24.9								

Table 7: Sparco problems with $\sigma = 0.001\|b\|_2$, (a) runtime in seconds, (b) relative duality gap, (c) number of root-finding iterations. The percentage reduction in time of the hybrid method over SPG is given in blue next to the problem index. Entries marked in gray indicate a solution that is not a root.

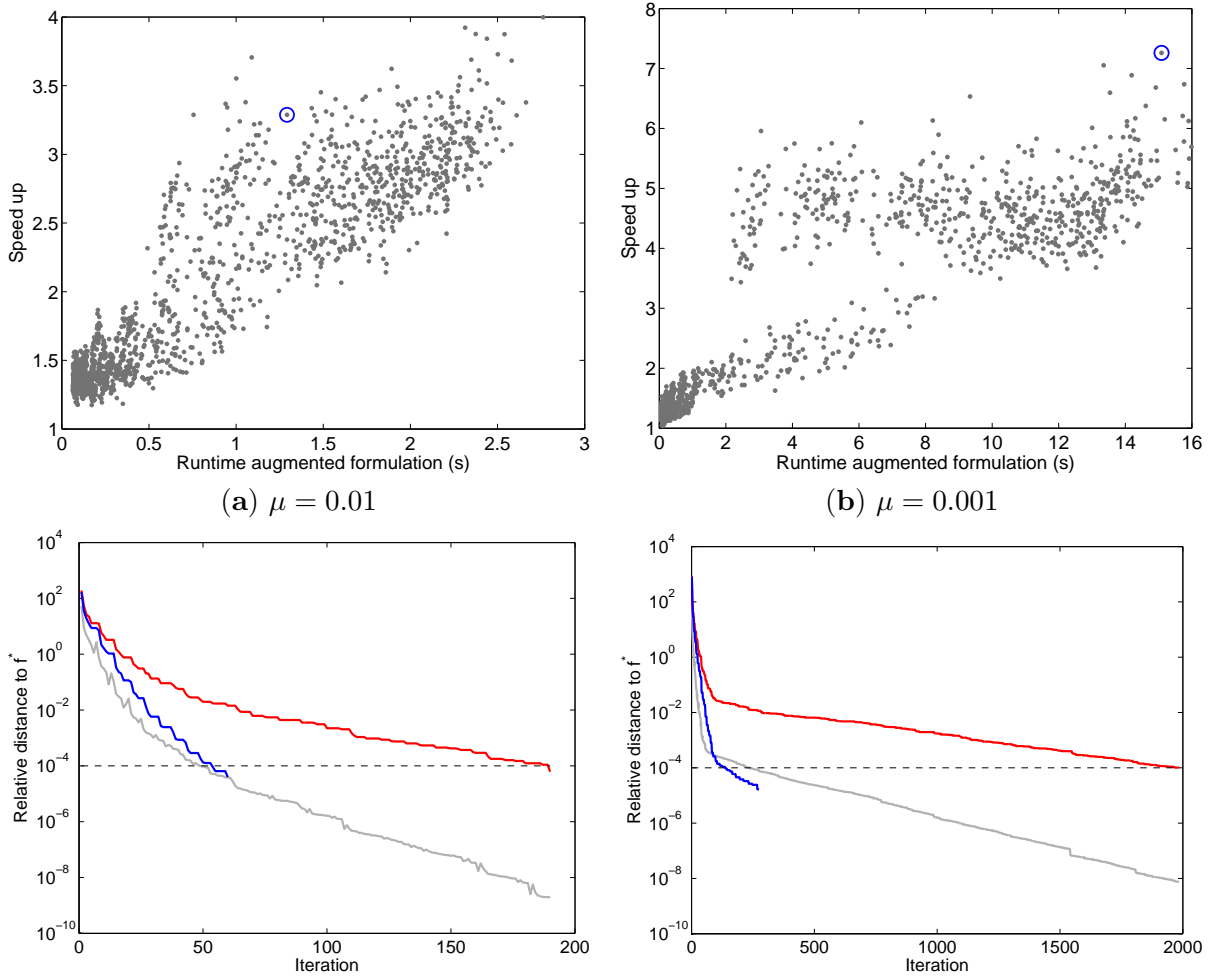


Figure 6: Plots of (a,b) the time required to reach the desired relative optimality gap of 10^{-4} using the augmented formulation, and the corresponding speed up obtained using the optimized formulation. Each point indicates a single problem instance; and (c,d) the relative distance of the primal (gray) and dual objective (red for the augmented formulation and blue for the optimized formulation) to the optimal objective as a function of iterations for the two circled problem instances.

and range μ log-linearly from 10^{-1} to 10^{-4} in four steps. As a result of the additive $\frac{\mu}{2}\|x\|_2$ term in the objective, the solutions are no longer sparse. As a result the hybrid method tends to coincide with the SPG method, and we therefore only consider the latter for these experiments.

For each of the settings we evaluate the time required by the augmented and optimized formulations to reach a relative duality gap of 10^{-4} . Figures 6(a,b) plots the speed up obtained using the optimized formulation against the runtime of the augmented formulation for two levels of μ . Despite the slightly more expensive evaluation of the dual, we see that the optimized formulation is around 1.5 to 4 times faster for $\mu = 0.01$, and up to 7 times faster for $\mu = 0.001$. For $\mu = 0.1$ (not shown in the plot) the speedup ranges from 1.2 to 3, and for $\mu = 0.001$ the speed up exceeds 10 on many problem instances and reaches a maximum of around 30.

We now take a closer look at the relative distance of the primal and dual objective to the optimum for the two circled problems in Figures 6(c,d). The progress of the primal objective over the iterations, indicated by the gray line, is the same for both formulations. For the dual objective there is a marked difference between the two. Notably, the augmented formulation

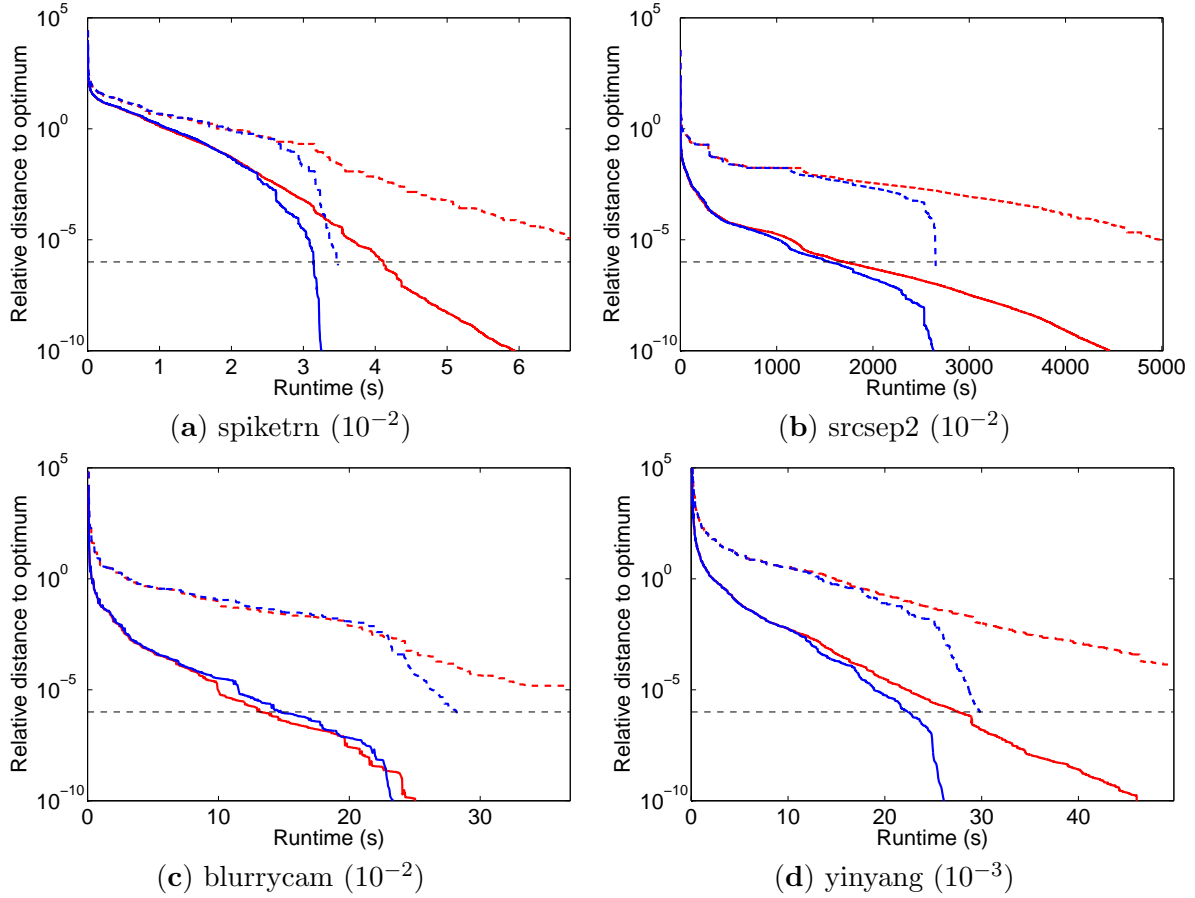


Figure 7: *Relative distance of primal (solid) and dual iterates (dashed) to the optimum as a function of time using the SPG (red) and hybrid method (blue) on four Sparco problems for fixed τ corresponding to the root for σ equal to the given multiple of $\|b\|_2$.*

converges much slower than the optimized formulation, thereby preventing the stopping criterion from being satisfied for many more iterations.

We now take another look at the Sparco problems from Tables 6 and 7. For each of the settings we record the optimal τ and then run the hybrid solver with a target optimality tolerance of 10^{-8} to obtain a best-effort optimum (for some problems the line search failed before reaching the desired tolerance). We then run the SPG and hybrid solvers with a target accuracy of 10^{-5} and record the relative distance of the primal and dual objective to the optimum at every iteration. The results for four representative problems are plotted in Figure 7. From the plots we see that the iterates of the hybrid method initially coincide or otherwise closely follow those of the SPG method. Once the hybrid method starts using quasi-Newton iterates increasingly often we see a sharp decrease in the relative distance to the optimum of the primal and dual iterates. The iterates of the SPG method, by contrast, continue to decrease very slowly. Indeed, of the fourteen problem settings, the SPG method managed to solve only two to the desired level of accuracy. Of the remaining problems, two reach the default iteration limit of ten times the number of rows in A , while all other problems fail with a line-search error. The hybrid method manages to solve all problems except for problem 401 with multiplier 10^{-3} . This problem reached the iteration limit, but could otherwise be solved successfully to a tolerance level of even 10^{-8} .

As before, we see that the dual objective converges to the optimum much slower than the

	$(\sigma = 0.01)$					$(\sigma = 0.001)$				
	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
701 - blurrycam	9.15	5.93	3.79	>3.66	>2.79	3.55	2.73	>2.21	>1.94	>1.61
702 - blurspike	2.83	2.39	2.16	>1.93	>1.66	3.22	2.48	2.20	2.07	>1.81
601 - soccer1	1.80	1.25	1.07	1.02	1.02	2.59	2.84	2.88	>2.57	>2.09
903 - spiketrn	1.62	1.64	1.73	>1.83	>1.63	1.45	1.53	>1.64	>1.56	>1.44
603 - yinyang	2.00	1.70	1.62	1.64	1.52	3.54	3.02	>2.76	>2.24	>1.81
401 - srcsep1	68.59	40.72	26.71	>9.49	>4.15	118.54	75.78	>49.75	>12.30	>2.38
402 - srcsep2	20.44	15.96	9.81	>4.35	>2.94	54.46	56.53	22.74	9.88	>2.78
(a) SPG										
701 - blurrycam	9.45	5.51	3.20	2.27	1.93	3.96	3.62	2.73	1.90	1.48
702 - blurspike	2.74	2.20	1.81	1.56	1.45	2.72	2.30	1.82	1.61	1.42
601 - soccer1	1.24	1.19	1.08	1.01	1.02	1.35	1.11	1.03	1.02	1.02
903 - spiketrn	1.34	1.23	1.14	1.09	1.11	1.19	1.13	1.13	1.11	1.09
603 - yinyang	2.23	1.80	1.64	1.55	1.45	2.99	2.12	1.63	1.52	1.33
401 - srcsep1	38.12	24.83	11.83	3.72	1.81	119.99	65.28	46.18	>13.44	>2.61
402 - srcsep2	17.46	11.62	6.45	2.60	1.72	50.98	44.70	16.92	6.07	1.99
(b) Hybrid										

Table 8: *Projected speed up when the optimal objective value is known and satisfaction of the optimality condition depends only on the primal objective value. We give a lower bound (indicated by the ‘>’ sign) when the dual objective failed to reach the given optimality level, either because the maximum number of iterations was reached, or because a line-search error occurred.*

primal, and unfortunately, there is no clear way to extend the optimized dual formulation from Section 4.3 to the standard Lasso formulation where $\mu = 0$. Given that the satisfaction of the optimality condition is controlled almost entirely by the dual objective value, it makes sense to look at the potential speed up if the optimal objective value was known and optimality was instead driven by the primal objective. In Table 8 we provide this speed up for the different Sparco problems with varying optimality tolerance levels. Clearly, both the SPG and hybrid methods would benefit from an improved dual, although the effect is less for the hybrid method, due to the already fast convergence of the dual objective in the final iterations.

6 Conclusions

In this paper we have presented a hybrid algorithm for minimization of quadratic functions over weighted one-norm balls. The method extends the spectral projected gradient method with L-BFGS iterations applied to reparameterizations of the objective function over active faces of the one-norm ball. For the decision of the iteration type we introduce the self-projection cone of a face and provide a complete characterization of this cone for weighted one-norm balls. The reparameterization uses an implicit orthonormal basis for the current face, and we provide an efficient algorithm for matrix-vector multiplication with this basis and its transpose. Our regular first-order iterations use backtracking line search of projected gradient steps. In addition to this we investigate the use of a trajectory line search over the entire projection curve of $x + \alpha d$ with $\alpha \geq 0$. We show that this curve is piecewise linear with at most $4n - 1$ segments, and that a local or global minimum of the objective along this curve can be determined efficiently. Despite this, the computational cost was still found to be high relative to projected backtracking line search, which showed overall better performance.

As part of the numerical experiments we propose a challenging class of test problems in which the columns of the $m \times n$ measurement matrix A are generated based on a random walk over the $(m-1)$ -sphere. Based on extensive numerical experiments on these and other test problems we showed that the hybrid method outperforms the original spectral projected gradient methods on a large fraction of the problems. Especially for medium to high accuracy solves and more challenging problems the SPG method was found to either take much more time to reach the desired level of accuracy, or fail prematurely due to line-search problems. The current stopping criterion of both methods relies on the generation of a dual feasible point from the primal iterate to determine the relative optimality of the iterate. From the experiments we found that the primal objective converges to the optimum much faster than the dual objective, and that satisfaction of the stopping criterion therefore depends entirely on the dual objective reaching the critical threshold. The performance of both methods could therefore be improved substantially given a better dual estimate.

In this paper we have studied the application of the hybrid method to the Lasso problem. Other important problems that may benefit from the approach but not discussed in this paper include box-constrained optimization and minimization of quadratic functions over the simplex.

A Proof of Theorem 4.3

In this section we study the combinatorial properties of the projection of the line $x(\alpha) = o - \alpha d$, onto an n -dimensional one-norm ball $\mathcal{C}_{w,1}$ of radius τ . Without loss of generality we can assume that the slopes d_i satisfy $d_i \geq 0$, by changing the signs of o_i and d_i if needed (this consistently reverses the signs of individual coordinates, but does not otherwise affect the projection). When plotting $v_i(\alpha) = |o_i - \alpha d_i|/w_i$ as a function of α we get horizontal lines when $d_i = 0$, and v-shaped curves when $d_i > 0$, as illustrated in Figure 8(a). Also plotted in this figure is $\lambda(\alpha)$: the soft-thresholding parameter λ used to project $x(\alpha)$ onto $\mathcal{C}_{w,1}$. For any α we can see that the support \mathcal{I} in the projection consists of all indices i for which $v_i(\alpha) > \lambda(\alpha)$. Over intervals of α where the support remains fixed, we have

$$\lambda'(\alpha) = \frac{\sum_{i \in \mathcal{I}} w_i r_i(\alpha)}{\sum_{j \in \mathcal{I}} w_j^2}, \quad r_i(\alpha) = \begin{cases} -d_i & o_i - \alpha d_i > 0 \\ d_i & \text{otherwise.} \end{cases}$$

Changes in the support occur whenever $\lambda(\alpha)$ intersects some $v_i(\alpha)$. Letting α' be a critical value where such an intersection occurs we can consider the linear segments of $\lambda(\alpha)$ that end, respectively start at this value of α . Within each of these segments we can choose arbitrary points $\alpha^- < \alpha$ and $\alpha^+ > \alpha$. For a single addition of entry i to the support we must have $r_i(\alpha^-)/w_i > \lambda'(\alpha^-)$, and it therefore follows from (26) that

$$\lambda'(\alpha^-) = \frac{\sum_{j \in \mathcal{I}} w_j r_j(\alpha)}{\sum_{j \in \mathcal{I}} w_j^2} < \frac{w_i r_i + \sum_{j \in \mathcal{I}} w_j r_j(\alpha)}{w_i^2 + \sum_{j \in \mathcal{I}} w_j^2} = \lambda'(\alpha^+).$$

For the removal of a single entry i from the support we must have $r_i(\alpha^-)/w_i < \lambda'(\alpha^-)$ it likewise follows that

$$\lambda'(\alpha^-) = \frac{\sum_{j \in \mathcal{I}} w_j r_j(\alpha)}{\sum_{j \in \mathcal{I}} w_j^2} = \frac{w_i r_i(\alpha) + \sum_{j \in \mathcal{I} \setminus i} w_j r_j(\alpha)}{w_i^2 + \sum_{j \in \mathcal{I} \setminus i} w_j^2} < \frac{\sum_{j \in \mathcal{I} \setminus i} w_j r_j(\alpha)}{\sum_{j \in \mathcal{I} \setminus i} w_j^2} = \lambda'(\alpha^+).$$

Multiple simultaneous changes to the support can be dealt with one at a time in a similar manner, resulting in $\lambda'(\alpha^-) < \lambda'(\alpha^+)$. When $x(\alpha) \in \mathcal{C}_{w,1}$ we have $\lambda(\alpha) = 0$. At the entry point we must

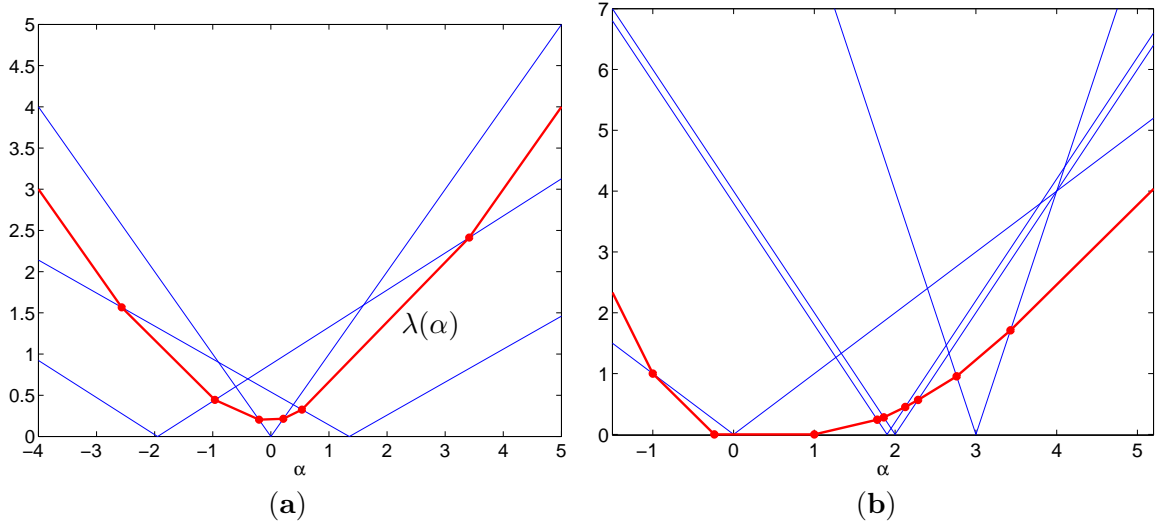


Figure 8: Plot of the absolute value of $x_i + \alpha d_i$ (blue curves) and soft-thresholding parameter $\lambda(\alpha)$ (red) as a function of α , with (a) a generic situation, and (b) a configuration for $n = 4$ that attains the maximum of $4n - 2 = 14$ intersections (five of these appear outside the plotted region).

have $\lambda'(\alpha^-) < 0 = \lambda'(\alpha^+)$, and for the exit point we have $\lambda(\alpha^-) = 0 < \lambda'(\alpha^+)$. Let slope $s_i = d_i/w_i$, $s_{\max} = \max_j s_j$, and k be any index such that $s_k = s_{\max}$, then for sufficiently negative α we have $v_k(\alpha) - v_i(\alpha) > \tau$ for all i such that $s_i < s_{\max}$. This implies that $[x_i(\alpha) - w_i \lambda(\alpha)]_+ = 0$, and therefore that only those entries with the maximum slope can be in the support, thus giving $\lambda'(\alpha) = -s_{\max}$. Similarly, we have $\lambda'(\alpha) = s_{\max}$ for sufficiently large α . Summarizing we have that $\lambda(\alpha)$ is piecewise linear with slopes strictly increasing from $-s_{\max}$ to s_{\max} , and we therefore conclude that $\lambda(\alpha)$ is convex.

A.1 Upper bound

From the convexity of $\lambda(\alpha)$ it immediately follows that the maximum number of intersections of $\lambda(\alpha)$ with any $v_i(\alpha)$ is four whenever $s_i < s_{\max}$ and two whenever $s_i = s_{\max}$. If $\lambda(\alpha)$ reaches zero there are two more break points, but between these points there must be at least one v_i that reaches zero, thereby removing two possible intersections with that curve. Since there is at least one index for which $s_i = s_{\max}$, the maximum possible number of break points is therefore $4(n - 1) + 2 = 4n - 2$. As each break point corresponds to a transition from one face to the next, it follows that the maximum number of faces of $\mathcal{C}_{w,1}$ that a line can project onto is $4n - 1$.

A.2 Constructions for the weighted one-norm ball

For $n = 1$ the maximum number of three faces is reached whenever $d_1 \neq 0$. For any $n \geq 2$ we can use the construction illustrated in Figure 8(b), consisting of two individual curves and a bundle of $n - 2$ curves in between. Working with zero crossings z_i instead of origin values o_i we define the first curve by $z_1 = 0$, $s_1 = 1$, and weight $w_1 = \omega$ to be specified later. The second curve has $z_2 = 3$, $s_2 = s_{\max} = 4$, and $w_2 = 1$. For each of the remaining $n - 2$ curves we sample the zero crossing z_k i.i.d. from $\mathcal{U}(1.9, 2)$, and choose $s_k = 2$ and $w_k = 1$. These values are chosen such that $v_i(4) \geq 4$. The only two parameters that remain to be chosen are ω and τ , and the approach is then as follows. By choosing ω sufficiently large, the minimum of $\|x(\alpha)\|_{w,1}$ occurs at $\alpha = 0$. We can then choose τ such that $\lambda(1) = 0$ forms a break point. Along with a second

zero crossing for some $\alpha < 0$ and additional intersections for sufficiently small and large values of α , this gives a total of four break points from the first curve. It then remains to ensure that $\lambda(4) < 4$, in which case we have two intersections of $\lambda(\alpha)$ with each of the remaining curves on the interval $\alpha \in [1, 4]$. The random sampling of z_k for $k \geq 2$ ensures with probability one that no three curves cross at the same point. For $k \geq 2$ we have $s_k < s_{\max}$, and each of these curves will therefore have an additional two intersections for sufficiently large positive and negative values of α . This gives the desired $4n - 2$ break points and $4n - 1$ faces.

The weighted one-norm $\|x(\alpha)\|_{w,1}$ can be verified to be equal to $\sum_i s_i |\alpha - z_i| w_i^2$. The directional derivative at $\alpha = 0$ is equal to $\omega^2 - 2n$, and for this to be positive we need to choose $\omega > \sqrt{2n}$. It now remains to choose an ω such that $\lambda(4) < 4$. A sufficient condition for this is that the directional derivative $\lambda'(\alpha) < 4/3$ for all $\alpha \in [1, 4]$. Initially we have $\lambda'(1) = \frac{\omega^2 - 2n}{\omega^2 + (n-1)} < 1$, and we must therefore first intersect a curve with index $i \geq 2$. At the first break point we have $\lambda' = \frac{c_1 + \omega^2}{c_2 + \omega^2}$ for some c_1 and c_2 , and λ' can therefore be kept smaller than $4/3$ by choosing a sufficiently large ω . This process can be repeated until we arrive at $\alpha = 4$ by taking the largest necessary ω over all steps. Using (26) we find that the largest combination of including or excluding each of the curves $i \geq 2$ is attained by including all indices, giving $\lambda' = \frac{2n + \omega^2}{(n-1) + \omega^2}$. Choosing $\omega > \sqrt{2n + 4}$ gives $\lambda' = \lambda'(4) < 4/3$, as needed. Figure 8(b) plots the result for $\omega = \sqrt{2n + 5}$.

A.3 Constructions for the canonical one-norm ball

For $n = 1$ the maximum number of three faces is reached whenever $d_1 \neq 0$. For $n = 2$ the construction in Figure 4 for a weighted one-norm ball attains the upper bound of seven. For the canonical one-norm ball, it is easily seen that the number of faces that $\lambda(\alpha)$ can project onto is at most five. An example construction that attains the maximum for $n = 3$ is plotted in Figure 9(a). The construction uses slopes $d = [1.0, 0.5, 1.0 - 10^{-3}]$, and offsets s such that the zero crossings are at $z = [-4, 0, 4]$. The slopes of the right- and left-most curves are chosen nearly identical to keep the sum of these components nearly constant for $\alpha \in [-4, 4]$. The middle curve has a smaller slope and forces the one-norm to grow from the middle outwards. Choosing $\tau = \|x - \alpha d\|_1$ at $\alpha = 3$ causes λ to be zero between $\alpha \approx -3$ and $\alpha = 3$. This is close enough to the point where the outer curves reach zero to ensure two intersections of $\lambda(\alpha)$ with each of these two curves before it intersects the middle curve again (intersections occur outside of the plotted range). The slope of the right-most curve is chosen slightly less than the maximum to ensure that it will eventually be intersected again by $\lambda(\alpha)$ at large positive and negative values of α .

A.3.1 Four dimensions

The construction of $n = 4$ uses parameters $d = [1.02, 0.52, 0.80, 1.01]$ with zero crossings at $z = [0.00, 0.21, 0.44, 0.86]$ and is plotted in Figure 9(b). Some of the intersections occur outside of the plotted range, but note that for curves with a slope that is smaller than the maximum, it suffices to have $\lambda(\alpha)$ below the curve on either side of two intersections; the slope of $\lambda(\alpha)$ eventually has to match the maximum of d and must therefore cross. The right-most curve has a slope that is just below the maximum slope and has its first two intersections at α near 0.75 and 2. The two intersections with left-most and steepest curve occur at α close to 0.05 and -1 .

A.3.2 Higher dimensional construction

For $n = 5$ we can use the general setup illustrated in Figure 10. It consists of two outer curves crossing at $-\mu_2$ and μ_2 with slopes 4 and $4 - \epsilon$, respectively, for some small $\epsilon > 0$. Next there is a

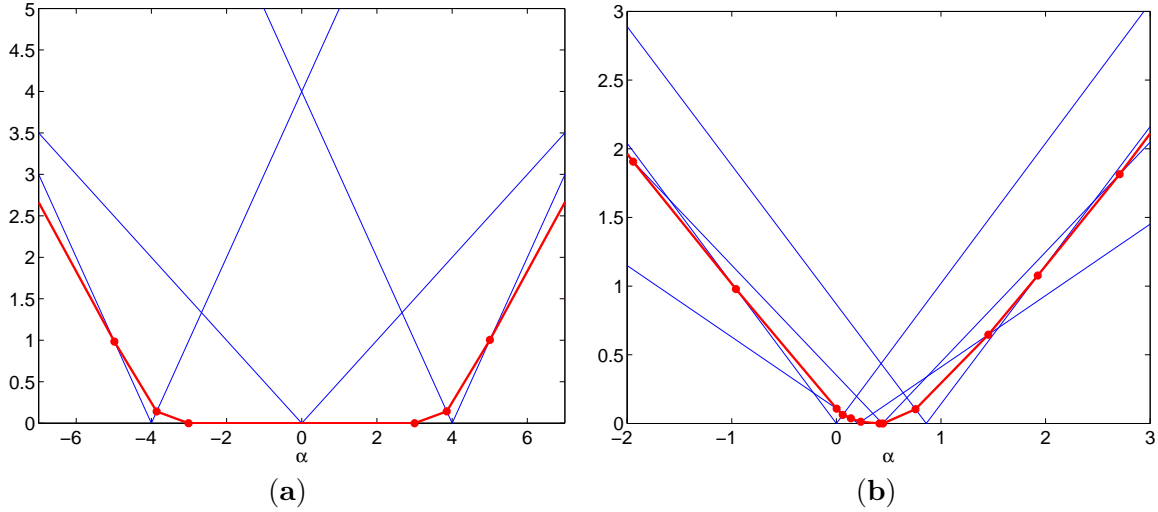


Figure 9: *Examples that attains the upper bound of $4n - 2$ intersections for (a) three dimensions and (b) four dimensions (some intersections occur outside the plotted region).*

bundle of $k_1 = \lfloor (n-3)/2 \rfloor$ uniformly sampled i.i.d. in the interval $[-\sigma, \sigma]$ around $-\mu_1$ with slopes 2. Another bundle of $k_2 = \lceil (n-3)/2 \rceil$ is uniformly sampled i.i.d. in the interval $[-\sigma, \sigma]$ around μ_1 , with slopes $2k_1/k_2$. Finally there is a central curve with zero crossing at 0 and slope 1. Aside from the $-\epsilon$ term (which is there to ensure that only one curve attains the maximum slope) and excluding the central curve, the one-norm remains constant between $-\mu_1 + \sigma$ and $\mu_1 - \sigma$. This enables us to force λ to be zero between $-\mu_1 + 2\delta$ and $\mu_1 - 2\delta$ and ensure a total of four break points for the central curve. By choosing δ and σ sufficiently close we then force two crossings of $\lambda(\alpha)$ with the curves in each of the two bundles (and an additional two crossings with each curve for sufficiently large positive and negative values of α). We then make sure to place μ_2 close to μ_1 such that the two outer curves too are intersected twice before crossing the central line again (this can be done by choosing μ_1 large enough and give the central curve enough space to grow sufficiently large). The proof is done in a number of steps:

- Step 1.** Derive conditions such that $\lambda(\alpha)$ changes to or from 0 at $-\mu_1 + \delta$ and in the interval $[\mu_1 - 2\delta, \mu_1 - \delta]$;
- Step 2.** Determine conditions on σ and δ under which $\lambda(\alpha)$ crosses all curves in each of the bundles and remains below them at a distance β from $\pm\mu_1$;
- Step 3.** Determine μ_2 relative to μ_1 such that $\lambda(\alpha)$ crosses the outer curves;
- Step 4.** The entire construction allows us to change μ_1 (and μ_2, ϵ accordingly) without changing the intersections of the bundles and outer curves (aside from minor effects due to ϵ). In the last step we therefore choose μ_1 to ensure that the central curve remains above $\lambda(\alpha)$ until after the outer two curves have been intersected.

We now consider each of the different steps.

Step 1 – zero crossing We choose τ such that $\lambda(\alpha)$ reaches 0 at $\alpha = -\mu_1 + \delta$. This is done simply by equation τ to the sum of the values of the curves at this point. Because $\epsilon > 0$ the sum of the curves at $\mu_1 - \delta$ will exceed τ and the zero crossing of $\lambda(\alpha)$ must therefore occur before this point. We now choose ϵ such that the one norm at $\mu_1 - 2\delta$ is no greater than τ . The contribution

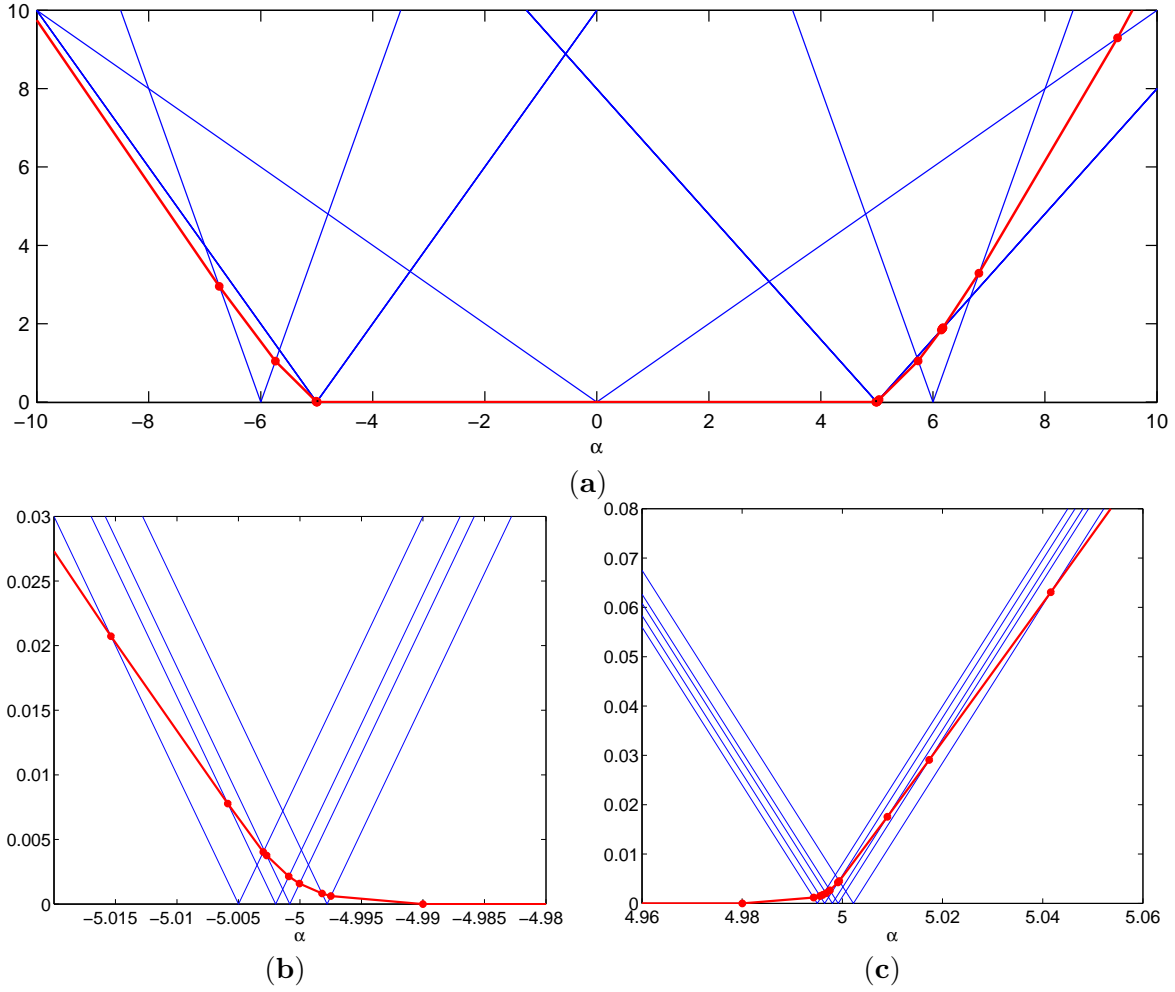


Figure 10: Plots showing (a) the configuration for $n = 12$, with $\beta = \frac{1}{2}$ and $\delta = 0.01$; and (b,c) the intersections of $\lambda(\alpha)$ with the center bundles around $\pm\mu_1$.

of the curves within each of the bundles affect the value of τ , but their contribution to the sum remains constant over the entire interval $[-\mu_1 + \sigma, \mu_1 - \sigma]$ and can therefore be ignored. Looking only at the relative difference we require that

$$(\mu_1 - \delta) \cdot 1 \geq (\mu_1 - 2\delta) \cdot 1 + (2\mu_1 - 3\delta)\epsilon,$$

which reduces to $\epsilon \leq \delta/(2\mu_1 - 3\delta)$, or the sufficient condition that $\epsilon \leq \delta/2\mu_1$.

Step 2 – crossing the bundles We analyze the crossing of the bundles (see illustration in Figures 10(b,c)) by taking the maximum slope over the entire path and start from $-\mu_1 + 2\delta$ on the left and $\mu_1 - 2\delta$ on the right (this causes the intersections to occur higher than they would otherwise). By choosing μ_1 large enough we can always ensure that the middle curve remains above all intersections. Aside from this, the results in this step are independent of μ_1 . We analyze the left and right bundles in turn, starting from the left bundle. The maximum relevant slope is directly to the left of the bundle and is equal to $(4k_1 + 1 - \epsilon)/n$, and we can use $(4k_1 + 1)/n$ for simplicity. We want the value of $\lambda(\alpha)$ to be below the bundle at $\alpha = -\mu_1 - \beta$ for some $\beta > \sigma$. This gives

$$(\beta - \sigma) \cdot 2 \geq (\beta + 2\delta) \cdot (4k_1 + 1)/n$$

or

$$(2n - 4k_1 - 1)\beta \geq 2\delta(4k_1 + 1) + 2n\delta$$

Choosing $\sigma = \delta/2$ and using the fact that $k_1 \leq (n - 3)/2$ gives the sufficient condition

$$(2n - 2(n - 3) - 1)\beta \geq 4\delta(n - 3) + n\delta,$$

which reduces to $\delta \leq 5/(5n - 3)\beta$, which certainly holds whenever $\delta \leq \beta/n$. Because the zero crossing of the curves within the bundle are chosen uniformly at random it holds with probability one that $\lambda(\alpha)$ does not cross at any of the intersections between the lines in the bundle.

For the right bundle we find a maximum slope of $(4k_1 + 1 + \epsilon)/n$, and to guaranteed $\lambda(\alpha)$ to be below the bundle at $\alpha = \mu_1 + \beta$ we require

$$(\beta - \sigma) \cdot 2k_1/k_2 \geq (\beta + 2\delta) \cdot (4k_1 + 1 + \epsilon)/n,$$

or, using $\sigma = \delta/2$,

$$(2nk_1/k_2 - (4k_1 + 1 + \epsilon)) \cdot \beta \geq 2\delta(4k_1 + 1 + \epsilon) + n(k_1/k_2)\delta. \quad (35)$$

For $k_1 = k_2$ we have $k_1 = (n - 3)/2$, and

$$(2nk_1/k_2 - (4k_1 + 1 + \epsilon))\beta = (2n - ((2n - 6) + 1 + \epsilon))\beta = (5 - \epsilon)\beta \geq 4\beta, \quad (36)$$

for $\epsilon \leq 1$. For the right hand side of (35) we use $2k_1 = n - 3$, and have

$$((8k_1 + 2 + 2\epsilon) + n(k_1/k_2))\delta = ((4n - 12) + 2 + 2\epsilon + n)\delta = (5n - 10 + 2\epsilon)\delta \leq 5n\delta \quad (37)$$

Combining (36) and (37) gives the sufficient condition $\delta \leq 4\beta/5n$. For the case where $k_2 = k_1 + 1$ we can use $k_1 = (n - 4)/2$, and $2n/k_2 \leq 6$. For the left-hand side of (35) this gives

$$\begin{aligned} (2nk_1/k_2 - (4k_1 + 1 + \epsilon))\beta &= (2nk_1/k_2 - (2n - 8 + 1 + \epsilon))\beta \\ &= (2nk_1/k_2 - 2n(k_1 + 1)/k_2 + 7 - \epsilon)\beta \\ &\geq (-2n/k_2 + 7 - \epsilon)\beta \geq (1 - \epsilon)\beta \end{aligned}$$

For the right-hand side of (35) we have

$$\begin{aligned} 2\delta(4k_1 + 1 + \epsilon) + n(k_1/k_2)\delta &\leq (8k_1 + 2 + 2\epsilon)\delta + 6k_1\delta \\ &= (14k_1 + 2 + 2\epsilon)\delta = (7n - 28 + 2 + 2\epsilon)\delta \\ &\leq 7n\delta \end{aligned}$$

This gives a sufficient condition of $\delta \leq \beta/8n$ for $\epsilon \leq 1/8$. This is the strongest condition of the three cases, and we can therefore choose $\beta = 8n\delta$.

Step 3 – crossing the outer curves First we have to find a minimum distance between μ_1 and μ_2 such that the outer curves are above $\lambda(-\mu_1 - \beta) \leq 2\beta$ and $\lambda(\mu_1 + \beta) \leq 2\beta k_1/k_2 \leq 2\beta$. Taking the smaller slope of $4 - \epsilon$ it suffices to have

$$(\mu_2 - \mu_1 - \beta) \cdot (4 - \epsilon) \geq 2\beta, \quad \text{or} \quad \frac{6 - \epsilon}{4 - \epsilon}\beta \leq \mu_2 - \mu_1.$$

For $\epsilon \leq 1$ it then suffices to have $(6/3)\beta \leq \mu_2 - \mu_1$, which is satisfied for $\beta = (\mu_2 - \mu_1)/2$.

We now consider the intersection of $\lambda(\alpha)$ with the outer curves, starting with the left (steepest) curve. Because we only need to intersect a single curve twice, we can use the maximum slope before the second intersection. The slope before the first crossing is less than 2 since we just intersected the bunch around $-\mu_1$. After the first crossing with the outer curve at $-\mu_2$ the slope changes to $(4k_1 + 1 + (4 - \epsilon))/(n - 1)$. For $k_1 = k_2$ we have $2k_1 = n - 3$ and

$$\frac{4k_1 + 1 + (4 - \epsilon)}{n - 1} = \frac{2n - 6 + 5 - \epsilon}{n - 1} = \frac{2(n - 1) + 1 - \epsilon}{n - 1} > 2,$$

whenever $\epsilon < 1$. This means that it can intersect the curves from the left bundle before the second intersection and for the maximum slope we therefore presume all of these curves are crossed, giving a slope of

$$\frac{2k_1 + 1 + (4 - \epsilon)}{k_2 + 2} = \frac{2(k_1 + 2) + 1 - \epsilon}{k_1 + 2} \leq 2 + \frac{1 - \epsilon}{k_1 + 2} \leq 2 + \frac{1}{3}.$$

For $k_2 = k_1 + 1$ we have $2k_1 = n - 4$ and

$$\frac{4k_1 + 1 + (4 - \epsilon)}{n - 1} = \frac{2n - 8 + 5 - \epsilon}{n - 1} = \frac{2(n - 1) - 1 - \epsilon}{n - 1} < 2.$$

This means that the curves from the bundle will not be intersected before the second intersection with the left-most curve.

For the right-most curve, we have a slope of $(4k_1 + 1 + 4)/(n - 1)$ directly after the first intersection. For $k_1 = k_2$ we can follow the same derivation as above (in this case with $\epsilon = 0$) to find that the maximum slope, after intersecting the curves from the right bundle is bounded by $2 + \frac{1}{3}$. For $k_2 = k_1 + 1$ we use $2k_2 = n - 2$ and find

$$\frac{4k_1 + 1 + 4}{n - 1} = \frac{4k_1 + 5}{n - 1} > \frac{4k_1}{n - 2} = 2\frac{k_1}{k_2},$$

so again, the curves from the right bundle can be intersected. Assuming all of them are intersected before the second intersection with the right-most curve we have a slope bounded again by $2 + \frac{1}{3}$.

We now take the maximum slope to be 3 and compute the maximum distance γ from $\pm\mu_2$ for which the second intersection with the outer curves occurs. It suffices to work with the right-most curve. We can start at $\alpha = \mu_1 + \beta$, the minimum curve of the right bundle can be below 2β , which immediately gives $\lambda(\alpha) \leq 2\beta$. Taken together we need to find γ such that

$$2\beta + 3(\mu_2 - (\mu_1 + \beta) + \gamma) \leq (4 - \epsilon)\gamma$$

Reorganizing gives

$$3(\mu_2 - \mu_1) - \beta \leq (1 - \epsilon)\gamma$$

With the choice of $\beta = (\mu_2 - \mu_1)/2$ this simplifies to $5(\mu_2 - \mu_1) \leq 2(1 - \epsilon)\gamma$. Since we chose $\epsilon \leq 1/8$ we have $\epsilon \leq 1/5$ and therefore it suffices to have $(\mu_2 - \mu_1) \leq 8\gamma$, which allows us to choose $\gamma = 4\beta$.

Step 4 – Controlling μ_1 We can satisfy all required inequalities thus far by fixing $\beta > 0$ and choosing $\mu_1 > 0$. In particular we can set $\mu_2 = \mu_1 + 2\beta$, and choose $\delta = \beta/n$. Since we already fixed $\sigma = \delta/2$ and $\gamma = 4\beta$, it suffices to choose $\epsilon = \min\{\delta/2\mu_1, 1/8\}$, which always gives $\epsilon > 0$. For the construction to work we must make sure that the central curve is above $\lambda(\alpha)$ for $\alpha = \pm(\mu_2 + \gamma)$. It suffices to be above the outer curves at this point and therefore

$$1 \cdot (\mu_2 + \gamma) \geq 4 \cdot \gamma.$$

Expanding γ and μ_2 gives $\mu_1 + 6\beta \geq 16\beta$, which is satisfied for $\mu_1 = 10\beta$. The plot in Figure 10 illustrates the construction for $n = 12$.

B Line search

B.1 First Wolfe condition

We want

$$f(x + \alpha d) \leq f(x) + \gamma_1 \alpha (\nabla f(x))^T d. \quad (38)$$

Expanding the left-hand side gives

$$\begin{aligned} f(x + \alpha d) &= \frac{1}{2} \|Ax - b + \alpha Ad\|_2^2 + \frac{\mu}{2} \|x + \alpha d\|_2^2 + c^T(x + \alpha d) \\ &= \frac{1}{2} \|r + \alpha Ad\|_2^2 + \frac{\mu}{2} \|x\|_2^2 + \mu \alpha x^T d + \frac{\mu}{2} \alpha^2 \|d\|_2^2 + c^T x + \alpha c^T d \\ &= \frac{1}{2} \|r\|_2^2 + \alpha r^T Ad + \frac{1}{2} \alpha^2 \|Ad\|_2^2 + \frac{\mu}{2} \|x\|_2^2 + \mu \alpha x^T d + \frac{\mu}{2} \alpha^2 \|d\|_2^2 + c^T x + \alpha c^T d \end{aligned}$$

For the right-hand side we have

$$f(x) + \gamma_1 \alpha (\nabla f(x))^T d = \frac{1}{2} \|r\|_2^2 + \frac{\mu}{2} \|x\|_2^2 + c^T x + \gamma_1 \alpha (A^T r + \mu x + c)^T d$$

To satisfy (38) we subtract the two

$$\begin{aligned} f(x + \alpha d) - f(x) - \gamma_1 \alpha (\nabla f(x))^T d &\leq 0 \\ \alpha r^T Ad + \frac{1}{2} \alpha^2 \|Ad\|_2^2 + \mu \alpha x^T d + \frac{\mu}{2} \alpha^2 \|d\|_2^2 + \alpha c^T d - \gamma_1 \alpha (A^T r + \mu x + c)^T d &\leq 0 \end{aligned}$$

Regrouping the terms gives

$$\begin{aligned} (1 - \gamma_1) \alpha r^T Ad + \frac{1}{2} \alpha^2 \|Ad\|_2^2 + \mu (1 - \gamma_1) \alpha x^T d + \frac{\mu}{2} \alpha^2 \|d\|_2^2 + (1 - \gamma_1) \alpha c^T d &\leq 0 \\ \alpha^2 (\frac{1}{2} \|Ad\|_2^2 + \frac{\mu}{2} \|d\|_2^2) + \alpha (1 - \gamma_1) \cdot (r^T Ad + \mu x^T d + c^T d) &\leq 0 \end{aligned}$$

Dividing by α and rearranging gives

$$\alpha \leq -(1 - \gamma_1) \frac{r^T Ad + \mu x^T d + c^T d}{\frac{1}{2} \|Ad\|_2^2 + \frac{1}{2} \mu \|d\|_2^2} = -2(1 - \gamma_1) \frac{(A^T r + \mu x + c)^T d}{\|Ad\|_2^2 + \mu \|d\|_2^2} = 2(1 - \gamma_1) \alpha_{\text{opt}}$$

B.2 Second Wolfe condition

We require

$$(\nabla f(x + \alpha d))^T d \geq \gamma_2 (\nabla f(x))^T d$$

Expanding gives

$$\begin{aligned} [A^T(r + \alpha Ad) + \mu(x + \alpha d) + c]^T d &\geq \gamma_2 (A^T r + \mu x + c)^T d \\ \alpha \|Ad\|_2^2 + \mu \alpha \|d\|_2^2 &\geq (\gamma_2 - 1) (A^T r + \mu x + c)^T d \\ \alpha &\geq -(1 - \gamma_2) \frac{(A^T r + \mu x + c)^T d}{\|Ad\|_2^2 + \mu \|d\|_2^2} \\ \alpha &\geq (1 - \gamma_2) \alpha_{\text{opt}} \end{aligned}$$

Acknowledgments

This work was supported by National Science Foundation Grant DMS 0906812 (American Reinvestment and Recovery Act). The implementation of the code uses L-BFGS update routines kindly provided by Michael P. Friedlander.

References

- [1] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.
- [2] Ewout van den Berg and Michael P. Friedlander. Probing the Pareto frontier for basis-pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008.
- [3] Ewout van den Berg and Michael P. Friedlander. Sparse optimization with least-squares constraints. *SIAM Journal on Optimization*, 21(4):1201–1229, 2011.
- [4] Ewout van den Berg, Michael P. Friedlander, Gilles Hennenfent, Felix J. Herrmann, Rayan Saab, and Özgür Yılmaz. Algorithm 890: Sparco: A testing framework for sparse reconstruction. *ACM Transactions on Mathematical Software*, 35(4):1–16, February 2009.
- [5] Dimitri P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, AC-21(2):174–184, April 1976.
- [6] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 2003.
- [7] Ernesto G. Birgin, José Mario Martínez, and Marcos Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.
- [8] Emmanuel Candès, Yonina C. Eldar, Deanna Needell, and Paige Randall. Compressed sensing with coherent and redundant dictionaries. *Applied and Computational Harmonic Analysis*, 31(1):59–73, 2011.
- [9] Emmanuel J. Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, February 2006.
- [10] Emmanuel J. Candès and Terence Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(2):4203–4215, December 2005.
- [11] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.
- [12] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006.
- [13] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the ℓ_1 -ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 272–279, 2008.
- [14] Roger Fletcher. A limited memory steepest descent method. *Mathematical Programming, Series A*, 135:413–436, 2012.
- [15] Branko Grünbaum. *Convex Polytopes*, volume 221 of *Graduate Texts in Mathematics*. Springer-Verlag, second edition, 2003.
- [16] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming, Series B*, 45(1–3):503–528, August 1989.

- [17] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, second edition, 2006.
- [18] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.